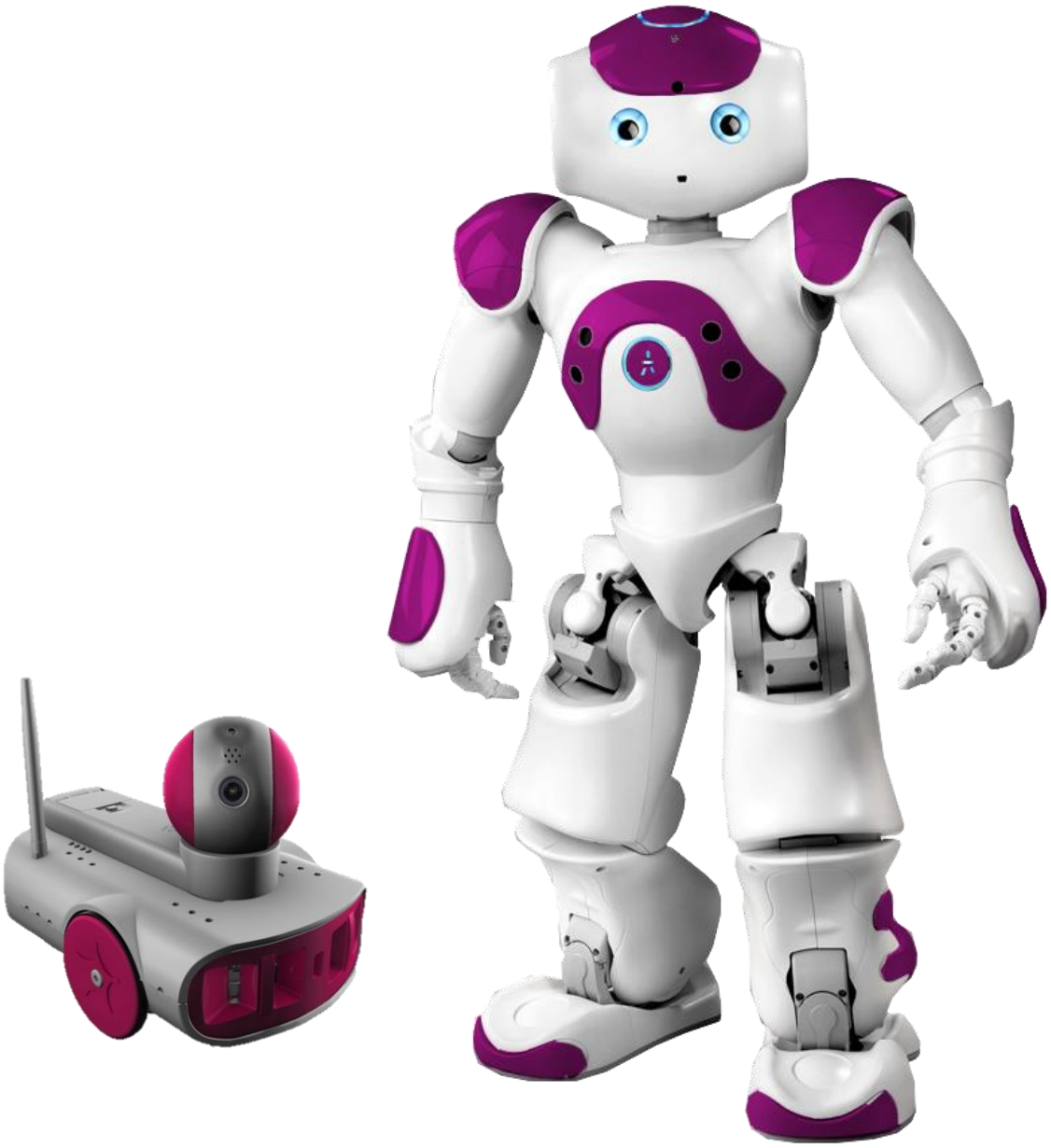
ARTS ET MÉTIERS ParisTech
Paris

Libre

CER PARIS 19/02/2013



DÉVELOPPEMENT DE MODULES DE VISION POUR LES ROBOTS NAO ET EMOX

RAPPORT DU PROJET FIN D'ÉTUDES

Del ARROYO ÁLVAREZ, LUIS
CANO GUEREÑA, DANIEL

Tuteur pédagogique : René BOULAIRE

ANNÉE : 2013 **N° du PE** :

CENTRE DE RATTACHEMENT PE : Arts et Métiers ParisTech CER Paris

AUTEURS : Luis Del ARROYO ÁLVAREZ / Daniel CANO GUEREÑA

TITRE : Développement de modules de vision pour les robots NAO et EMOX.

ENCADREMENT DU PE : M. René Boulaire. Laboratoire de Mécatronique et Robotique

NOMBRE DE PAGES :69 **NOMBRE DE RÉFÉRENCES BIBLIOGRAPHIQUES** : 50

RÉSUME : Développement d'une application pour la lecture de codes QR pour Nao. Recherche des différents codes existants et son intégration sur le robot.
Présentation du nouveau robot Emox, description, installation et premiers pas. Développement d'un module pour le suivi d'une ligne codée par différentes couleurs. Solution des problèmes trouvés.

MOTS-CLÉS : PROGRAMMATION ROBOTIQUE / NAO / ALDEBARAN / CODE QR / DATAMATRIX / CODE DE BARRES / NAOMARK / CHOREGRAPHE / QRDECODER / ZBAR / OPENCV / EMOX / AWABOT / EMOXSOFT / SUIVI LIGNE / DÉTECTION COULEUR.

ACCÈS AU DOCUMENT :

PARTIE À REMPLIR PAR LE PROFESSEUR RESPONSABLE DU PROJET
ACCESSIBILITÉ DE CE RAPPORT (entourer la mention choisie) :
Classe 0 = accès libre
Classe 1 = Confidentiel jusqu'au _ _ _ _ _ _ _ _ _
Classe 2 = Hautement confidentiel
Date : Nom du signataire : Signature :

Remerciements :

Tout d'abord nous tenons à remercier l'ENSAM et notre tuteur de projet, M. Boulaire, qui nous ont donné l'opportunité d'avoir un contact direct avec la mécatronique et qui nous ont confié l'accès à deux robots différents. Nos amis de l'université: MaximeDuprés, Fantiago, Zeva, Mordiscain, LeRate, Guarritrónica, Bully, Naseto, Poio, Nïniö, Pichi, Francis, Electroperra, Feli, Alpert, Metrallote, Acho et à l'équipe Bang : LanderLanda, XabiRed, VictorDisney et Evita.

De mon côté, Luis, je remercie mes parents, mon frère Íñigo, Mayte qui a été toujours avec moi, même en France, et mes amis de l'école et de l'université qui m'ont accompagné tout au long de mes études d'ingénierie pendant six ans et deux pays différents.

De ma part, Daniel, je souhaite remercier toute ma famille, nire neskita, et mes amis du lycée qui m'ont toujours accompagné : Le Caf, Momo, Olmet, Pato et 12, Jant, Mr Ricard, Pinny et chan chan, Il Barone, Truñette, Gererare,Tilde et Barbie, Fanti, Cubano...

SOMMAIRE

PRÉSENTATION DU PROJET FIN D'ÉTUDES 5

1. INTRODUCTION 7

1.1. ÉTAT DE L'ART 7
1.1.1. Nao 7
1.1.2. Emox 7
1.1.3. Codes en 1D et 2D 9

2. NAO 11

2.1. INSTALLATION DES OUTILS ET DES LIBRAIRIES NÉCESSAIRES POUR LA LECTURE AVEC NAO SUR LINUX UBUNTU. 11

2.2. LES CODES ÉTUDIÉS 12
2.2.1. NaoMarks 12
2.2.2. Code de barres 15
2.2.3. Code QR 16
2.2.4. Datamatrix 17

2.3. LES LIBRAIRIES ET LES APPLICATIONS ÉTUDIÉES 18
2.3.1. qrdecoder ..... **¡Error! Marcador no definido.**
2.3.1.1. PyQT4, PIL et SIP 19
2.3.2. ZBar ..... **¡Error! Marcador no definido.**
2.3.2.1. Usage d'OpenCV au lieu d'ImageMagick 21
2.3.3. OpenCV ..... **¡Error! Marcador no definido.**
2.3.4. libdmtx (projet Datamatrix) 23
2.3.5. ImageMagick 24
2.3.6. Autres librairies étudiées 25

2.4. INTÉGRATION DES APPLICATIONS SUR NAO 25
2.4.1. Par console 25
2.4.2. Choregraphe 29

3. EMOX 32

3.1. DESCRIPTION TECHNIQUE 32

3.2. INSTALLATION ET CONNEXION 39

3.3. PROGRAMMATION DU ROBOT 44

3.4. SUIVI D'UNE LIGNE 47
3.4.1. Possible amélioration du hardware 51

3.5. DÉTECTION DE LA COULEUR 53
3.5.1. Comment obtenir les valeurs des couleurs 54
3.5.1.1. À travers d'une application externe 54
3.5.1.2. À travers de la console de debug d'Emox Soft 56

3.6. SUIVI D'UNE LIGNE AVEC CODE COULEUR 57

3.7. SOLUTIONS DE PROBLÈMES 62
3.7.1. Connexion voie router 62
3.7.2. Des autres problèmes trouvés 64

4. CONCLUSION 66

BIBLIOGRAPHIE 67

Présentation du projet fin d'études

Aujourd'hui, nous assistons à un monde qui est de plus en plus technologique. En effet, celle-ci est de plus en plus présente dans nos vies. Parmi l'expansion des nouvelles technologies, la mécatronique et la robotique jouent un rôle de plus en plus important et semblent destinés à s'intégrer dans le monde quotidien.

Dans ce cadre de développement, l'ENSAM veut y participer et un projet de mise en place d'un nouveau laboratoire de robotique à l'ENSAM Paris est en cours. L'école a donc acheté de nouveaux robots pour la recherche et la création des nouveaux modules avec plusieurs buts dans différents domaines, devenant ainsi un centre important dans le monde de la robotique.

Grâce à notre projet, nous avons participé à la mise en place de ce nouveau pôle, en démarrant un des premiers projets. Ce travail permettra d'augmenter la connaissance sur ce champ et donner de l'expérience pour les futurs projets. Notre recherche s'est poursuivie avec deux robots différents : Nao d'Aldebaran et Emox d'Awabot. Tous les deux sont de récentes acquisitions de l'école, plus précisément Nao a été acheté en février 2012 et Emox en janvier 2013. Après l'achat de Nao et, par conséquent, notre proposition de projet fin d'études, M. Boulaire a connu Emox dans le salon Innorobo à Lyon en mars. Il s'est renseigné sur le projet EmoxLab, sur les compétences du robot jusqu'au moment mais aussi sur des possibles projets à faire. Grâce à cette recherche, il a décidé de l'incorporer au laboratoire.

Les plateformes de notre projet étant Nao et Emox, nous avons cherché à développer des projets qui s'adaptent aux mieux à ses caractéristiques. Un des aspects les plus importants, est que ce sont de robots conçus pour l'interaction avec des humains. En effet, Nao étant humanoïde on s'adapte à lui d'une manière plus naturelle. Quant à Emox, c'est son apparence en tel qu'animal de compagnie qui lui confère la capacité de se rapprocher des humains de façon confortable. De plus, les capacités particulières de chaque robot ont déterminé nos études.

D'une part, Nao, est particulièrement adapté à l'aide à la personne, comme le montrent les projets d'Aldebaran sur des sujets tels que l'autisme. Nous avons donc décidé de chercher un projet qui permette Nao de soutenir les personnes âgées. À cet effet, nous nous sommes intéressés à la prise de médicaments, l'idée étant de lire des codes placés sur ces derniers afin de les identifier. Ceci réduira les soucis causés par la détérioration de la vue. La lecture de codes est un moyen très effectif de communiquer avec un robot puisqu'elle permet de transmettre une information de façon précise et sans erreurs. De plus, la lecture de codes n'est pas

un projet limité à un cas d'application mais est extrêmement versatile. Par exemple, on pourrait envisager un cas d'application où un robot identifie des personnes grâce à un badge avec un code, ou bien le code peut devenir une indication qui permet au robot de se situer dans un endroit. Donc, dans la réalisation de notre projet, nous travaillerons sur la lecture de codes avec Nao pour recevoir de l'information d'une manière fiable grâce à la vision, qui lui sera utile pour aider les personnes en identifiant des objets, des noms ou tout type de message.

D'autre part, Emox, est un robot qui est beaucoup plus mobile que Nao. Nous nous sommes donc proposé d'utiliser cette mobilité et de lui donner un sens. Imaginons, que le robot doit se déplacer à l'intérieur d'une usine. Afin d'arriver à sa destination, Emox doit s'orienter et connaître le chemin. Pour cela, une façon fiable d'y arriver est de suivre une ligne sur le sol qui arrive jusqu'au lieu indiqué, comme s'il s'agissait d'un train sur des rails. Cependant, nous cherchons à améliorer ce système, en rajoutant la détection de couleur au suivi de la ligne. Grâce à cette modification, Emox peut prendre des décisions différentes selon la couleur de la ligne, comme par exemple une modification de la vitesse (pour freiner avant un virage ou accélérer sur une ligne droite) ou bien suivre un chemin ou un autre selon la couleur.

Nous allons donc centrer notre projet sur la lecture de codes avec Nao et le déplacement d'Emox, en suivant un chemin avec des indications simples à travers de l'usage de couleurs.

1. Introduction

1.1.État de l'art

1.1.1. Nao

Le robot NAO est développé par Aldebaran Robotics, entreprise française 100% jusqu'à mars 2012 où un partie (plus de 80%) a été achetée par un groupe japonais appelé Japan's Softbank. L'entreprise japonais a payé 100M $ pour cette partie, et elle a investi entre 40M$ et 50M$ dans le projet.

NAO est un robot humanoïde ressemblant à un enfant, de 57 cm, avec 25 degrés de liberté programmable sur plusieurs langages telles que C++, Python, Urbi, .NET, Java et Matlab. Son cœur est un processeur Intel ATOM 1,6ghz CPU qui fonctionne sous un noyau Linux qui support l'intergiciel d'Aldebaran Naoqi

Notre projet est centré sous l'aspect vision du robot. Les caméras sont positionnées sur la tête (mises sur le front et sur la bouche), capables de prendre 30 images par second en 920p.

Aldebaran propose une banque de données où on trouve des projets en open source développés par des personnes ayant un accès dans le site. On trouve cet outil de projets dans developer.aldebaran-robotics.com.

Dans ce site nous avons trouvé un projet en relation au notre pour la lecture de codes DataMatrix [2]. La dernière version du projet est un fichier Choregraphe (DataMatrix-1.0.crg voir annexes) prêt pour le lancer. Pour la lecture des codes, le développeur a utilisé la librairie libdtmx [3] qui décode les DataMatrix et les codes de barres sur plusieurs systèmes d'exploitation. Le code est développé pour l'ancienne version de Nao, soit la 1.12. Version sur laquelle le programme a été démarré sans soucis. Malheureusement nous avons trouvé des problèmes de compatibilité avec la dernière version de Nao, classifiée 1.14.

1.1.2. Emox

Emox est un petit robot ressemblant à un chiot avec deux roues, une puissante caméra et plusieurs capteurs. Il mesure 23 cm de longueur 15,5 de largeur et 16 cm d'hauteur. À ce moment, Emox est un robot en développement, raison pour laquelle il est aussi appelé EmoxLab. Il est désigné par Awabot Robotics, une petite entreprise situé en Villeurbanne qui est surtout centrée dans ce projet.

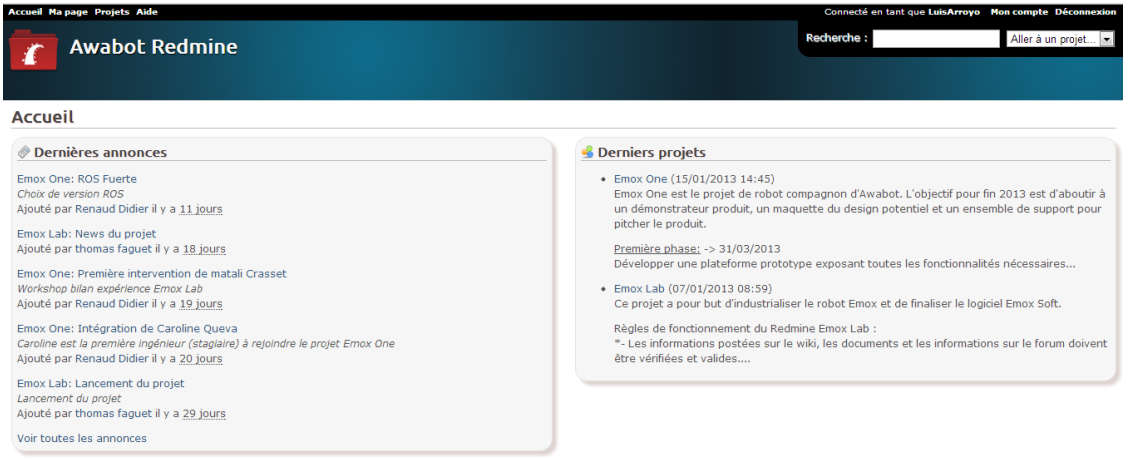
Comme on a expliqué, Emox n'est pas encore un robot commercialisé, il est vendu comme un outil pour la recherche. Dans l'image ci-dessous, les deux versions d'Emox sont présentées. À gauche, on trouve celle qui est vendue actuellement pour la recherche. C'est un prototype, il a juste le structure nécessaire mais il est équipé avec tous les éléments techniques nécessaires. À son coté, il est présenté le projet final qu'Awabot lancera au marché. Ils ont montré dans une vidéo les caractéristiques qu'Emox aura telle qu'un robot domestique :

http://www.youtube.com/watch?v=4eBqVn3uS5w

Dans cette vidéo il est montré comme le robot est capable de projeter des images pour, par exemple, aider aux enfants pendant ces jeux de construction en projetant les manuels et les tutoriaux, ou faire une vidéoconférence étant donné qu'il a une caméra très puissante et une connexion wifi.

Pendant une conférence Skype avec M. Faguet, il nous a confirmé qu'ils travaillent sur une nouvelle version d'Emox plus performante avec plusieurs améliorations mécaniques, qui sera présenté officiellement dans le salon Innorobo de robotique à Lyon du 19 au 21 mars 2013 (www.innorobo.com). Après cette date, Awabot nous changera le robot pour cette nouvelle version gratuitement.

D'autre part, Awabot dispose d'un forum afin d'y réunir à tous les personnes travaillant dans un projet d'Emox. Le site est appelé Awabot Redmine car il est basé sur l'outil de gestion de forums Redmine. On trouve ce site sous l'adresse : http://awabot-redmine.herokuapp.com/. En tant qu'utilisateur extérieur sans compte enregistrée, on peut accéder à quelques informations utiles sur le projet connu comme Emox One. Lorsqu'on crée une compte, on peut lire en plus les informations et les questions postées dans le forum sur le projet Emox Lab, ce que nous intéresse dans le cadre de notre projet.


Accueil Ma page Projets Aide
Connecté en tant que LuisArroyo Mon compte Déconnexion
Awabot Redmine
Recherche :
Aller à un projet...
Accueil
Dernières annonces
Emox One: ROS Fuerte
Choix de version ROS
Ajouté par Renaud Didier il y a 11 jours
Emox Lab: News du projet
Ajouté par thomas faguet il y a 18 jours
Emox One: Première intervention de matali Crasset
Workshop bilan expérience Emox Lab
Ajouté par Renaud Didier il y a 19 jours
Emox One: Intégration de Caroline Queva
Caroline est la première ingénieur (stagiaire) à rejoindre le projet Emox One
Ajouté par Renaud Didier il y a 20 jours
Emox Lab: Lancement du projet
Lancement du projet
Ajouté par thomas faguet il y a 29 jours
Voir toutes les annonces
Derniers projets
Emox One (15/01/2013 14:45)
Emox One est le projet de robot compagnon d'Awabot. L'objectif pour fin 2013 est d'aboutir à un démonstrateur produit, un maquette du design potentiel et un ensemble de support pour pitcher le produit.
Première phase: -> 31/03/2013
Développer une plateforme prototype exposant toutes les fonctionnalités nécessaires...
Emox Lab (07/01/2013 08:59)
Ce projet a pour but d'industrialiser le robot Emox et de finaliser le logiciel Emox Soft.
Règles de fonctionnement du Redmine Emox Lab :
"- Les informations postées sur le wiki, les documents et les informations sur le forum doivent être vérifiées et valides....

Ce projet appelé Emox One, c'est le pas suivant à l'Emox Lab. L'idée d'Awabot est de créer un robot plus performant à la fin de mars 2013, une maquette ressemblant au produit final avec laquelle ils pourront montrer les fonctionnalités souhaitées pour la version livrable qu'ils espèrent avoir fini à la fin de 2013.

1.1.3. Codes en 1D et 2D

La lecture de codes est très développée dans le domaine des applications mobiles pour les Smartphones. C'est la raison pour laquelle on trouve plusieurs librairies destinées à ce but, la plupart open source.

Aujourd'hui Il existe deux types de codes selon sa dimension, en 1D et 2D. Les codes en 1D sont les classiques codes de barres qu'on trouve dans la plupart de produits mais ils sont très limités. Même s'il y a plusieurs versions de ce type de codes, sa capacité est très petite.

Par rapport aux codes en 2D (ou Bidi, de bidimensionnelle), on trouve différents types mais nous nous allons centrer sur ceux qui nous concernent, les codes QR et les Datamatrix.

Les Datamatrix sont les codes trouvés, par exemple, sur les boites de médicaments mais ils ne sont pas très étendus. D'autre part, les codes QR ont des avantages par rapport à eux. Ils ont une taille minimale plus petite donc l'utilisation de l'espace est plus efficace, ils peuvent stocker des symboles japonais appelés symboles kanji et ils ont une meilleur gestion pour la perte d'information, donc il existe la possibilité d'y mettre au-dessus des logos. Cette particularité fait que ces codes- ci puissent porter des publicités et soient donc plus attractifs pour les entreprises.

Ci-dessous un code QR qui contient un lien vers http://www.ensam.fr/ :

2. NAO

2.1. Installation des outils et des librairies nécessaires pour la lecture avec Nao sur Linux Ubuntu.

Premièrement nous avertissons qu'il est très importante de regarder l'information en relation au système d'exploitation choisi dans le site de documentation d'Aldebaran, dans la rubrique *C++ SDK Installation* : http://www.aldebaran-robotics.com/documentation/dev/cpp/install_guide.html, car selon la version installée sur Nao on doit choisir une version concrète du SE, surtout par rapport à l'installation de Linux (normalement la dernière version d'Ubuntu ne sera pas compatible) car c'est un SE toujours mis à jour et il peut y avoir des incompatibilités avec le robot. On doit bien regarder sur le coin supérieur gauche que la documentation lue et celle de la version de Nao qu'on a mise en place. Le lien ci-dessus vous rédigera sur le site de l'installation du SDK pour la dernière version de Nao existante (actuellement 1.14.1).

Que ce soit Windows ou Ubuntu le SE sélectionné, c'est impératif de choisir la version 32 bits (au moins pour l'instant) car la compilation n'est pas encore fiable sur des systèmes 64 bits.

L'avantage de choisir Ubuntu par rapport à Windows c'est que dans le premier nous avons la possibilité de compiler du code directement sur le robot ou sur l'ordinateur, tandis que sur une machine Windows on ne peut que le faire sur l'ordinateur.

Nous avons choisi donc la version 10.04 32bits d'Ubuntu trouvable sur le site officiel où on peut y trouver les anciennes versions : http://old-releases.ubuntu.com/releases/

Une fois installé le SE et les outils nécessaires pour la compilation de Nao (information disponible sur le lien vers la documentation d'installation du SDK C++) il faut installer les dépendances des librairies qu'on utilisera.

Le cas dont nous parlerons c'est l'installation des dépendances pour l'outil que finalement nous avons utilisé, le qrdecorer, mais la démarche sera toujours pareille.

Au début on doit nous diriger au site de l'application souhaitée et chercher les librairies externes à installer. Dans notre cas, il est dit dans le site http://code.google.com/p/qrdecoder/ :

- Python - by the Python Software Foundation
- PyQt4 GUI toolkit by Phil Thompson - GNU GPLv3

- SIP by Phil Thompson - GNU GPLv3
- ZBar by Jeff Brown - GNU LGPLv2.1
- Python Imaging Library (PIL) by Secret Labs AB & Fredrik Lundh

Il existe un outil Linux pour faire l'installation de façon presque automatique. On doit ouvrir une terminale pour y travailler. Les commandes qu'on utilisera sont :

- `sudo apt-get install <librairie à installer>` : grâce à ce commande-ci le programme cherchera s'il existe la librairie dans la base de données, vous dira la taille de l'installation et vous demandera si l'on veut continuer ou pas. S'il n'existe pas la dépendance sous le nom que nous avons mis, il faut écrire le suivant commande.

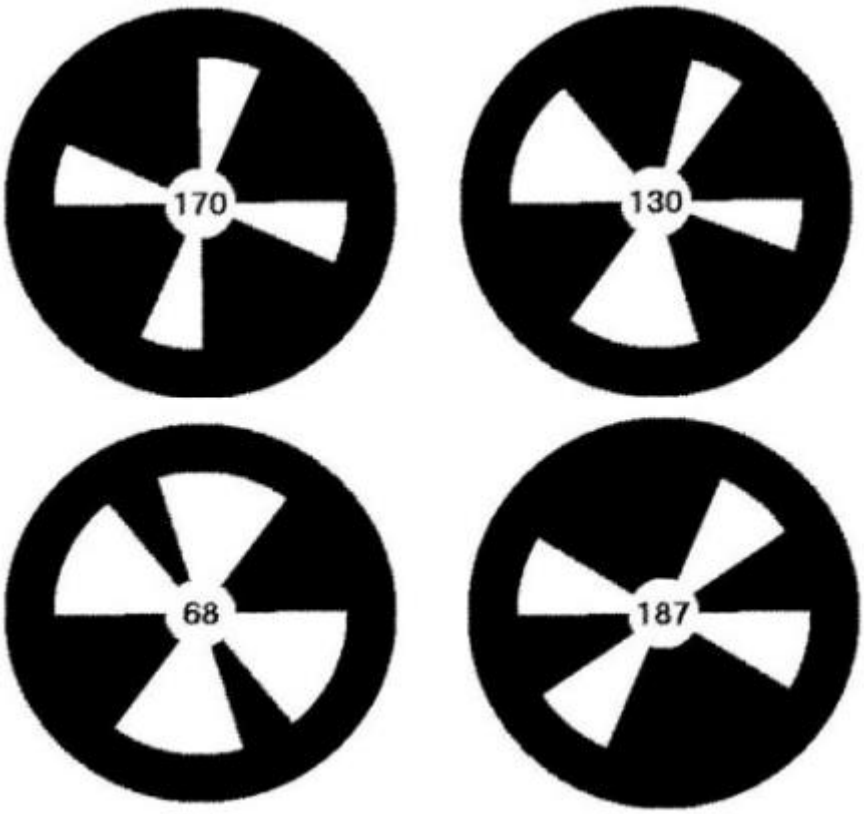
- `apt-cache search <librairie à chercher>` : parfois, lorsqu'on met le nom de la librairie l'application dit que n'existe pas. Ce cas n'est pas rare et c'est parce que il y a des librairies qui ont plusieurs paquets différents, plus d'une version, ou tout simplement parce que le nom n'est pas concret en soi-même. Si on trouve ce souci, on écrit `apt-cache search` et nous arriverons à trouver une liste des possibles solutions. On choisit celle qu'on croie qui sera la librairie cherchée, et on tape la commande d'installation ci-dessus.

Remarque : Il est nécessaire d'écrire le mot `sudo` devant la commande puisque on a besoin des droits de super-utilisateur (de même façon qu'on fait sur Windows en choisissant *en tant qu'administrateur*).

2.2.Les codes étudiés

2.2.1. NaoMarks

Lors de notre recherche de codes à lire par Nao, tout d'abord, nous avons commencé par vérifier s'il existait déjà un système intégré sur la plateforme. Cet ainsi que nous avons découvert les NaoMarks. Il s'agit de marques crées par Aldebaran qui contiennent des chiffres. Ce sont des cercles noirs, avec des triangles blancs orientés vers un cercle blanc central. La position des triangles blancs permet de distinguer les différentes NaoMark.

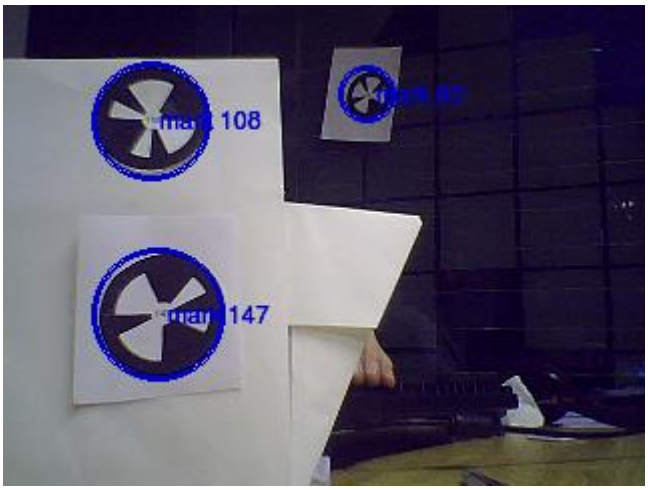


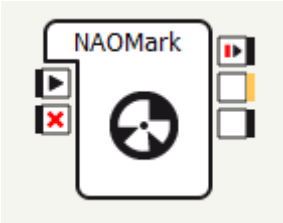
Les NaoMarks sont reconnues par le module ALLandMarkDetection, implémentée de base sur le robot. Ces signaux permettent de donner des petites informations au robot. Grâce au numéro identifié, Nao peut s'orienter ou exécuter une action particulière. Cependant cette information est très limitée puisqu'il ne s'agit que de numéros.

La lecture de Naomarks a été testée dans des conditions d'illumination d'entre 100 et 500 lux. Pour reconnaître les codes il est important que les images aient un fort contraste puisque que c'est justement le contraste de couleurs qui est analysé. Il est conseillé d'ajuster le contraste de la caméra pour une meilleure détection.

De plus, les NaoMark ont des contraintes en taille et en inclinaison. Ainsi, la taille de la marque doit être comprise entre 14 et 160 pixels sur une image QVGA (320x240 pixels). Dans le cas de l'inclinaison de l'image, pour pouvoir détecter la NaoMark correctement, celle-ci ne peut avoir que 60º d'inclinaison par rapport à la caméra. Par contre, un avantage des NaoMark est que sa rotation n'a aucune influence sur la détection.

Nous avons donc testé la fonctionnalité des NaoMarks. Premièrement, nous avons essayé avec le Monitor et le Choregraphe :





Comme on peut observer, sur l'image du monitor, Nao reconnaît automatiquement les NaoMarks dès qu'elles se trouvent dans son champ de vision. Avec le Choregraphe, il suffit d'ajouter le box pour lancer la détection.

Cependant, nous avons remarqué que la détection est difficile. En effet, les marques doivent se trouver près du robot dans des conditions de lumière où il n'y ait pas de reflets et le plus de face possible, au risque de que Nao commette une erreur et décode mal l'information.

Ensuite nous avons écrit notre propre module en C++, en utilisant AL::ALLandMarkDetection. Mais il y avait des difficultés à accéder à l'information et de savoir si la marque avait été réellement détectée. Or, comme nous ne voulions pas travailler exclusivement avec les NaoMarks puisqu'il s'agit d'un codage limité et non performant, nous n'avons plus continué sur ce code (voir AL::ALLandMarkDetection en annexe).

Finalement, nous avons donc rejeté les NaoMarks à cause de leur limité capacité d'information, la difficulté à les décoder puis la complexité de sa programmation en C++.

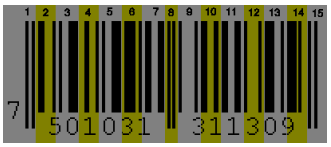
2.2.2. Code de barres

Les codes de barres sont des codes d'une dimension très étendus dans les marchés puisque :

- ils sont simples à faire et à imprimer
- ils peuvent stocker des numéros
- ils sont faciles à lire avec des lecteurs simples et pas chers

Les nouvelles versions des codes permettent aussi de stocker de petits textes mais le répertoire des caractères permis est très limité. D'autre part, ce n'est pas possible d'y mettre des gros textes.

Comme résultat de son expansion, aujourd'hui on trouve une vaste gamme d'encodages différents pour les codes de barres. Voici une liste des codes de barres lus par ZBar [6] :

Version	Capacité	Type de caractère	Exemple	Norme international e
EAN-13/UPC-A	Les UPC-A et EAN-13 sont les codes les plus utilisées. Ils sont ceux que normalement on trouve sur les produits du marché et des magasins.	Numérique		ISO/IEC 15420
UPC-E	Ces codes sont une version des codes UPC-A mais plus compacts font en éliminant les zéros contenus.	Numérique		

EAN-8	Petit code créé pour des petits colis	Numérique		ISO/IEC 15420
Code 128	Cette version permis d'encoder des caractères alphanumériques avec en plus d'une chiffre checksum pour vérifier l'information	Alphanumérique		ISO/IEC 15417
Code 39	Code alphanumérique standard	Alphanumérique		ISO/IEC 16388
Interlea-ved 2 of 5	Code très étendu dans le domaine de la distribution et le storage	Numérique		ISO/IEC 16390

2.2.3. Code QR

Créés en 1994 par Denso-wave, ces codes ont permis l'introduction des symboles Kanji japonais, ce que n'est pas possible avec les codes de barres ou les Datamatrix. Comme il a été expliqué dans l'introduction, les codes QR ont trois avantages principaux par rapport aux autres codes :

- **Son utilisation de l'espace est plus efficace**

L'information stockée sur la même aire que les autres codes est plus grande. Cela fait que l'espace occupé par le code soit mineur mais, en revanche, au fur et à mesure que l'information est plus grande, le code est plus dense et par conséquent la lecture devient plus difficile. Aujourd'hui, les caméras des Smartphones ont une puissance suffisante pour les lire, mais la résolution des caméras de Nao est plus faible, donc pour l'instant nous avons essayé avec de codes pas complexes.

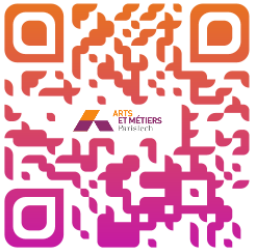
- **Le stockage des caractères japonais (symboles kanji)**

À cause de l'internationalisation des technologies, une standardisation des codes devienne nécessaire. Les codes Datamatrix ont été créés par une

entreprise nord-américaine, raison pour laquelle ils n'ont pas développé cet aspect. En revanche, les codes QR ont la possibilité de les emporter ce qui fait d'eux des codes plus internationaux.

- **Système de control de pertes plus efficace:**

L'algorithme de correction s'appelle Reed-Solomon. Cette correction permet d'avoir plusieurs niveaux d'erreur: 7, 15, 25 ou 30%. Le code enregistre l'information redondante afin d'avoir une compensation des bits qui seront perdus. Dans l'exemple exposé ci-dessous, nous avons fait un code QR pour accéder au site internet de l'école, mais nous avons mis le logo AM au-dessus de lui-même avec une petite distorsion des carrés. En plus, on peut changer la couleur de code sans soucis. L'outil on-line utilisé s'appelle Unitaglive: http://www.unitaglive.com/qrcode



2.2.4. Datamatrix

Ce troisième code étudié est en concurrence avec les codes QR. Il fut Inventé en 1989 par l'entreprise états-unienne International Data Matrix Inc. (aujourd'hui appelée CI Matrix). La dernière et plus utilisée version du code est datée du 2005.

Malgré que cette façon d'encryptage soit plus ancienne, elle n'a pas encore la possibilité d'y intégrer les codes japonais kanji. Dans le monde de la technologie, Japon fait une partie très importante, raison pour laquelle les codes QR sont nés et ont devenu les concurrents les plus forts.

Aujourd'hui, la guerre entre QR et Datamatrix n'est pas encore finie puisque selon ce qu'on veut faire l'un peut être mieux que l'autre. Pour les seconds, si on utilise des codes avec quatre aires différenciées et grâce à la correction d'erreur Reed Salomon nous pouvons aussi mettre ci-dessus des logos. Le souci qu'on trouve dans ce cas est que la densité de points est plus grande et donc on aura besoin des caméras plus puissantes. La caméra avec laquelle Nao est équipé n'a pas la qualité d'image nécessaire pour pouvoir lire aisément ces codes quand ils sont éloignés ou tournés. Le code sans le logo a été grâce au site http://datamatrix.kaywa.com/ :

Les codes Datamatrix sont surtout utilisés dans le monde industriel de la production. On peut les trouver, par exemple, dans les boites de médicaments ou dans les étiquettes collées sur les colis.

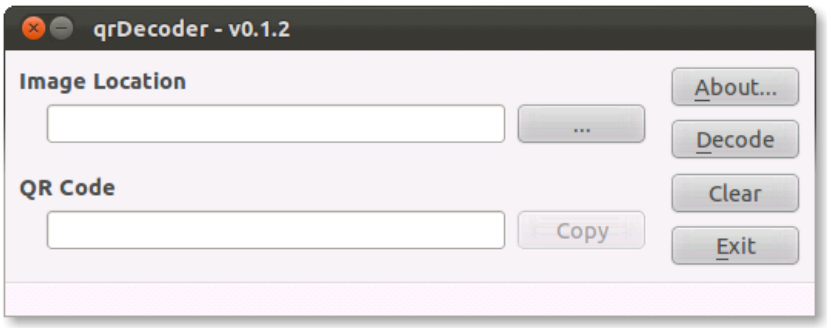
2.3.Les librairies et les applications étudiées

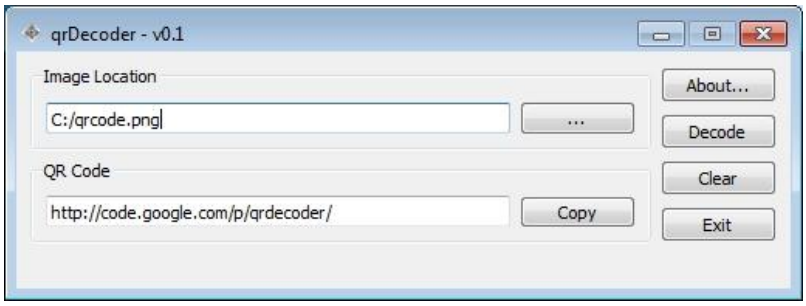
2.3.1. qrdecoder

qrdecoder es une application développée par un particulier appelé Nicholas Wilde, elle est open source et trouvable dans le site de Google de codes : http://code.google.com/p/qrdecoder/

Le principal but de l'application est la lecture de codes QR dans les images gardées dans l'ordinateur. L'application demandera l'adresse du fichier contenant le code QR et en suite, en cliquant sur le bouton ***Decode*** le message codé sera affiché. On a maintenant le choix de copier le texte ou de l'effacer et décoder une autre image. Les formats d'image supportés sont :

- PNG
- GIF
- TGA
- JPG
- PCX
- BMP
- TIF





L'utilisation est très simple, pouvant être démarré sous Windows ou Linux. Elle est basée principalement sur le toolkit PyQt pour le design de la fenêtre d'interaction, et les librairies ZBar pour la lecture des codes et PIL pour le traitement des images. À son tour, ZBar dépend d'autres librairies telles qu'ImageMagick.

D'autre part, cette application est programmée en Python. Pendant les cours de mécatronique suivis à l'ENSAM, M. Mechbal nous a conseillé d'utiliser ce langage puisque est très étendu dans le monde de la mécatronique, donc nous nous sommes renseignés sur cette mode de programmation. Une autre avantage de Python est la simplicité d'écriture ce qui fait de lui un code très intuitif. Étant donné que nous voulions changer et reprogrammer le code pour notre propre but, toutes ces raisons nous ont amenés à travailler avec le *qrdecoder*.

2.3.1.1. PyQT4, PIL et SIP

De ces trois librairies, celle que nous intéresse au niveau de notre projet est la librairie PIL. PyQT4 et la librairie des codes chargés de l'interface de l'application. La fenêtre affichée, les interactions de l'utilisateur avec les boutons, le design, etc. sont faits avec ce code. D'autre part, SIP (*Session Initiation Protocol*) est une librairie qui gère les sessions interactives avec des éléments multimédia utilisateur - machine.

Par contre, la librairie PIL (*Python Imaging Library*) fut créée pour le traitement des images dans les environnements Python. Elle est utilisée dans le *qrdecoder* quand on veut obtenir les données de l'image souhaitée. Le code concerné est :

```
# obtain image data
pil = Image.open(unicode(self.lineEdit_location.text())).convert('L')
width, height = pil.size
raw = pil.tostring()
```

Grâce à la première ligne on sauvegarde dans la variable *pil* quelques informations de l'image. Pour savoir ce qu'il est sauvegardé, on peut ajouter la ligne :

```
print pil
```

après de la première ligne et ensuite ouvrir une terminale Linux et y taper la commande suivante (après d'on se placer dans le dossier contenant le fichier) :

```
$ python qrdecoder.pyw
```

Cela démarrera le code et montrera ce que la variable *pil* contient, par exemple :

```
<Image.Image image mode=L size=250x300 at 0x93F18EC>
```

On voit que le mode de l'image est « L ». Cela est le produit d'avoir insérer la commande *convert('L')*. *Convert* est une méthode d'une instance Image qui retourne une copie transformée de l'image. Dans ce cas, l'ordre L transforme l'image de couleur à une image noir et blanc. La commande `self.lineEdit_location.text()` envoie l'adresse de l'image qu'on a mise dans l'interface.

La commande *pil.size* retourne les deux valeurs des dimensions de l'image : largeur et hauteur (250 et 300 dans l'exemple) :

```
width, height = pil.size
```

La troisième ligne (`raw = pil.tostring()`) sauvegarde dans *raw* une chaîne contenant les données des pixels de l'image avec un encodage de type « raw »

2.3.2. ZBar

ZBar est une librairie écrite en C open source, compatible avec CC+, python ainsi que d'autres langages de programmation. Cet outil, permet une rapide détection ainsi que le décodage de codes-barres et QR codes. Plus précisément, les codes que ZBar est capable de reconnaître sont :

- QR codes
- UPC-E
- Code 39
- EAN-13
- EAN-8
- Interleaved 2 of 5
- UPC-A
- Code 128

Avec cette librairie, on peut scanner des images une à une (fonction ImageScanner) ou bien une vidéo avec la mise en place d'un callback

(fonction The Processor). Des codes en C++ ou Python sont proposés pour utiliser ces fonctions :

http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=HOWTO:_Scan_images_using_the_API

http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=HOWTO:_Scan_video_using_the_Processor

Le scan de vidéos en temps réel, permis par ZBar, est très intéressant pour notre projet, puisque de cette façon Nao serai capable de détecter un code QR à n'importe quel instant, sans avoir à l'avertir par exemple.

Nous avons écrit un code en Python pour lire des QR codes dans des images qui se trouve en annexes. Ce code a été testé et fonctionne correctement sous linux.

2.3.2.1. Usage d'OpenCV au lieu d'ImageMagick

ZBar, n'est pas capable par lui seul d'ouvrir des fichiers, tel que de images, c'est pour cela que l'on doit faire recours à l'ImageMagick.

Cependant, Nao incorpore déjà une bibliothèque pour traiter des images. Il s'agit de l'OpenCV. De plus, dans la version 1.14, la version d'open cv intégrée dans Nao est upgradée à la 2.3.1.

Travailler avec ImageMagick nous oblige à inclure une bibliothèque supplémentaire, c'est pour cela qu'il est intéressant de remplacer ce logiciel par l'OpenCV. Pour faire ce remplacement, il faut faire charger l'image de la manière suivante en C++:

Avec ImageMagick

```
// read an image file

Magick::Image magick(argv[1]);

// extract dimensions

int width = magick.columns();

int height = magick.rows();

// extract the raw data

Magick::Blob blob;

magick.modifyImage();

magick.write(&blob, "GRAY", 8);

const void *raw = blob.data();

// wrap image data

Image image(width, height, "Y800", raw, width * height);

// scan the image for barcodes

int n = scanner.scan(image);
```

Avec OpenCV

```
CvMat*cv_matrix=cvLoadImageM(argv,CV_LOAD_IMAGE_GRAYSCALE);

ImageScanner scanner;

scanner.set_config(ZBAR_NONE,ZBAR_CFG_ENABLE, 1);

int width = cv_matrix->width;

int height= cv_matrix->height;

char*raw = (char*)cv_matrix->data.ptr

Image image(width, height, "Y800", raw, width * height);

int n = scanner.scan(image);
```

Nous n'avons pas finie la conversion du code, puisqu'il faudrait faire ce changement sur Python. La partie à changer en Python est la suivante :

```
# obtain image data
pil = Image.open(img).convert('L')
width, height = pil.size
raw = pil.tostring()

# wrap image data
image = ZBar.Image(width, height, 'Y800', raw)

# scan the image for barcodes
scanner.scan(image)
```

2.3.3. OpenCV

OpenCV est la librairie de vision open source la plus utilisé actuellement. Elle fut inventée par Intel en 1999 et aujourd'hui elle dispose de plus de 500 fonctions différentes compatibles sur GNU/Linux, Windows ou Mac OS X. Nao utilise ces librairies pour, par exemple, le module de reconnaissance de visages.

Dans le cadre de notre projet, nous avons eu besoin de cette librairie pour l'usage d'une autre librairie appelé ZBar dont nous parlons quelques sections après.

Le souci que nous avons eu c'est que normalement jusqu'à la version 2.1, OpenCV était divisé par librairies dans différentes domaines mais tous ensembles dans un module unique. Pour intégrer ces fonctions à notre code il n'a fallu qu'écrire:

```
#include "OpenCV\cv.h"
#include "OpenCV\highgui.h"
using namespace cv;
```

Mais c'est à partir de la version 2.2 qu'ils ont développé une sorte de « OpenCV 2 ». Depuis cette version la librairie est plus divisée en modules (qui sont: core, imgproc, video, calib3d, features2d, objdetect, highgui, gpu, et plus d'autres petits) et des autres changements. Maintenant, la façon de l'intégrer est :

```
#include "OpenCV2/core/core.hpp"
#include "OpenCV2/highgui/highgui.hpp"
using namespace cv;
```

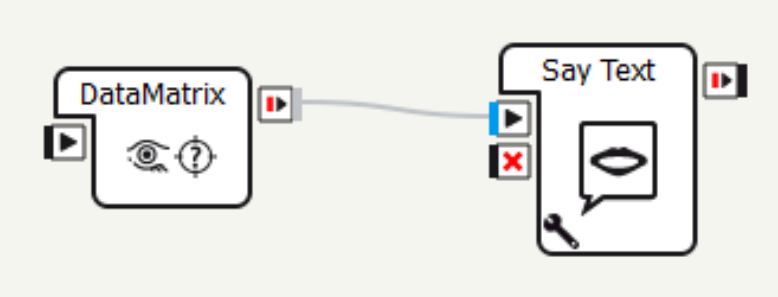
Nous avons constaté ce problème-ci quand nous avons mis à jour le système de Nao de la version 1.12 à la 1.14. Avec la 1.12 l'OpenCV compatible était le 2.1, mais à partir de la nouvelle version il utilise la 2.3 et les codes que nous avions écrits ne fonctionnaient plus, même pas le lecteur des codes Datamatrix téléchargé. Avant de pouvoir corriger le souci, nous avons pris connaissance que nous allons recevoir la nouvelle plateforme Emox donc nous n'avons plus pu nous consacrer à Nao pour le résoudre.

2.3.4. libdmtx (projet Datamatrix)

Libdmtx est une bibliothèque open source pour le codage et décodage de Data Matrix, pour Linux, Windows, etc.

Cette librairie a été choisie par Aldebaran pour son projet Datamatrix. Ce projet fut créé le 03-11-2010 par une équipe de développeurs d'Aldebaran et

modifié pour la dernière fois en 07-2012. Nous avons eu accès à ce projet grâce au site d'Aldebaran pour développeurs. La dernière version du projet consiste en un fichier Choregraphe. Pour ajouter la librairie au projet, libdtmx a été linké en statique au projet.



Nous avons testé l'application sur Nao avec succès sur la version 1.12. Malheureusement, le projet n'est plus valable sur la version 1.14 de Nao, qui est actuellement sur le robot. Lors des tests nous avons placé des images avec des Data Matrix devant Nao. Ce dernier les a décodé, puis nous a transmis l'information vocalement. Cependant, le décodage prenait beaucoup de temps, pouvant arriver à une vingtaine de secondes. De plus, nous avons essayé de faire une démonstration devant un groupe d'élèves en PJ M mais nous n'avons pas réussi que Nao lise les Data Matrix placés devant lui. Ce test nous a montré que Nao n'est pas encore très au point pour ce type de défis. En effet, après quelques nouveaux essais, nous pouvons expliquer que pour une lecture des codes, ceux-ci doivent se trouver près de la caméra et être suffisamment grands et droits par rapport au robot. De la même façon, les conditions de lumière sont très importantes, car s'il y a des reflets, la lecture devient quasi impossible. On peut diminuer les reflets en montrant au robot des images sur papier et non des images sur des écrans de PC ou similaire. On peut donc conclure, que le projet DataMatrix fonctionne correctement sur la version 1.12, mais qu'il s'agit d'un projet avec des limitations à cause des problèmes avec la qualité de l'image captée par le robot.

2.3.5. ImageMagick

ImageMagick est un ensemble de fonctions utilisables à partir de la ligne de commande et ne possède pas d'interface graphique, d'où son choix. En effet il est souvent utilisé dans des codes et est compatible avec plusieurs formats dont le C++ et le Python. Ce logiciel permet de manipuler des images sous une grande variété de formats. Parmi ses possibilités, on retrouve la modification des couleurs, le changement de format, la transformation d'images, la rotation, etc.

2.3.6. Autres librairies étudiées

Nous avons aussi révisé deux autres librairies décodant des codes QR que finalement nous n'avons pas retenu principalement parce qu'elles utilisent plusieurs dépendances externes et ces codes sont plus difficiles à modifier que celui de qrdecoder. Cet ajout de bibliothèques compliquait en excès la programmation et son usage sur Nao le serait également. Celles sont l'application ZXing et la librairie libdecoderqr.

2.4.Intégration des applications sur Nao

Nous avons essayé d'implanter nos propres modules pour le décodage de QR codes sur Nao, en utilisant la librairie ZBar. Pour cela nous avons utilisé deux méthodes :

- En créant des modules C++.
- Avec un projet Choregraphe.

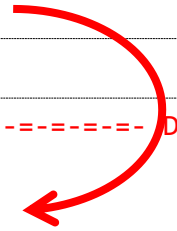
2.4.1. Par console

Prenant de basse le code *qrdecoder.pyw*, nous l'avons reprogrammé afin de pouvoir l'utiliser pour la reconnaissance des codes QR par Nao. Nous rappelons que le code original permet de décoder un code QR contenu dans une image qu'on ajoute dans l'interface de l'application. Ce que nous voulons développer, c'est de modifier le code afin de décoder l'image captée par Nao. Pour cela, ce que nous avons modifié est la façon de prendre le fichier. Normalement le code obtienne l'image à partir de l'adresse de cette façon-ci :

```
pil = Image.open(unicode(self.lineEdit_location.text())).convert('L')
```

Étant `self.lineEdit_location.text()` la commande qui retourne la localisation de l'image. Ici, ce qu'il faut changer c'est cette commande par un variable contenant l'adresse par elle-même. Autrement dit, ce qui Nao fait quand il prend des images est las sauvegarder en mémoire. Nous simulons le fonctionnement avec une adresse de fichier mais elle sera valide même avec une adresse de mémoire. Ci-dessous deux petites parties du code qui montrent son fonctionnement (voir le code complet en annexe 4. Decoder QR code avec ZBar) :

```
#=-=-=-=-=-=-=-=-=-=-=-=-=-=- MAIN  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

print "\n Quelle image voulez-vous décoder? \n"
print "\n"

imagen = raw_input (">")
#imagen = '/home/luis/PJE_linux/qrdecoder/qrdecoder-0.1.2/test/baboon.jpg'

decodeImage(imagen)
```



```
#=-=-=-=-=-=-=-=-=-=-=-=-=-=- DEF  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

def decodeImage(img):

    scanner = ZBar.ImageScanner()
    scanner.parse_config('enable')

    pil = Image.open(img).convert('L')
    width, height = pil.size
    raw = pil.tostring()

    pil.show()
```

Dans le main (il est positionné à la fin du code) nous définissons la variable *imagen* qui sera appelé dans la fonction decodeImage(). La fonction *raw_input (">")* permet obtenir l'adresse mise par terminale. Quand la fonction *decodeImage* est appelée, la variable d'entrée est *img*. En plus nous avons ajouté `pil.show()` à la fin, une fonction du PIL pour montrer l'image traitée.

Avec ces changements et l'élimination des lignes concernant l'interaction avec l'interface, le code est prêt pour recevoir l'image, la décoder et envoyer le message contenu. La partie du code chargée du décodage est :

```
# scan the image for barcodes
    scanner.scan(image)
    # extract results

    for symbol in image:
      if symbol:
          print "Image contain a code."
      else:
          print "Image does not contain a QR code."

    for symbol in image:

      print symbol.type

      # do something useful with results
      if symbol.type is not symbol.QRCODE:
          print "Image does not contain a QR code."
          return

      print "L'information du code est: \n ", symbol.data, "\n"

    # clean up
    del(image)
```

Dans un premier temps du test, ce que nous voulions faire est un code unique avec lequel on reçoive une image captée par Nao, cette image devienne la variable *img*, et le texte du code est envoyé à Nao. Pour les images, nous avons développé un code C++ permettant de capturer des images en cliquant la touche « espace » :

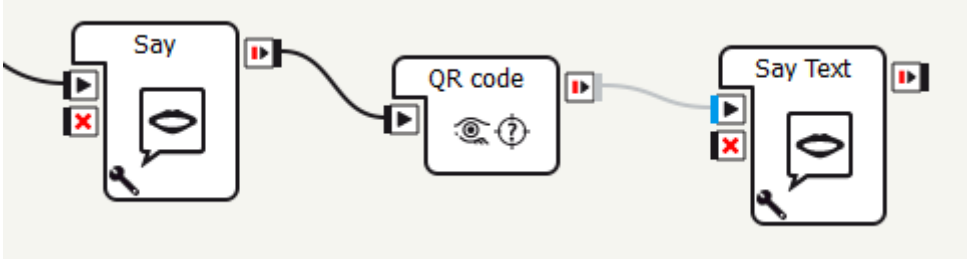
```
  /** Main loop. Exit when pressing ESC.*/
 // printf("\n aqui \n");
  bool loop=true;
  while (loop==true)
  {
         int select =(char) cv::waitKey(1000);
          switch(select)
                     {

                     case 27:
                            loop=false;

                     break;

                     case 32:
                                /** Tells to ALVideoDevice that it can give back the
image buffer to the
                                   * driver. Optional after a getImageRemote but
MANDATORY after a getImageLocal.*/
                              //cvReleaseImageHeader(&imgHeader);
                              //printf("\n bucle \n");
                              camProxy.releaseImage(clientName);
                              img = camProxy.getImageRemote(clientName);
                              imgHeader.data = (uchar*) img[6].GetBinary();
                              camProxy.releaseImage(clientName);
                              cv::imshow("images", imgHeader);
                              //cvSaveImage("imgnao.png", "images");
                              printf("done \n");

                     break;

                     default:
                            ;
                     };

  }
```

Malgré le qrdecoder est écrit en Python et le code qui prend les images en C++, cela ne représente pas un souci au niveau de compilation puisque Nao est capable de démarrer des codes en *cross-compilation* (seulement sur Linux).

Nous sommes arrivés jusqu'à ce point car la plateforme Emox est arrivée et nous avons dû arrêter le projet Nao. Le projet donc reste ouvert pour les nouveaux élèves.

2.4.2. Choregraphe

Après avoir vu le projet DataMatrix d'Aldebaran, nous avons voulu le modifier, de façon à ce qu'il lise les codes QR.



Pour cela, nous avons modifié le script bu box du projet d'Aldebaran. Le script, récupère l'image du robot, puis l'analyse avec le *libdtmx*. C'est cette dernière partie qu'il faut modifier.

```
   # Get image from Nao's camera

self.cam.subscribe("dmtx", kQQVGA, kRGBColorSpace, 5); # Warning, use
QQVGA (160x120px) otherwise it's horribly long

 img = self.cam.getImageRemote("dmtx");

 self.cam.unsubscribe("dmtx");
```

Pour décoder l'image, on utilise le même code que dans la section précédente (*par console*) :

```
# Decode the image

decodeImage(img) ;  ## au lieu de self.dm.decode(img[0], img[1],
                        buffer(img[6]));

## On rappelle que  img = self.cam.getImageRemote("nom_image");

    # Parse results

    result = list(symbol.data);

    ##pour remplacer result = list(self.dm.message(i) for i in range(1,
    self.dm.count()+1));
```

Une fois ce code écrit, nous devons inclure la librairie dans le projet. C'est cette étape qui n'a pas été conclue, et qui ne nous a donc pas permis de tester nos codes.

Nous avons essayé de linker les librairies en les copiant dans un dossier Choregraphe comme l'avaient fait les développeurs d'Aldebaran dans une version antérieure.

```
def onLoad(self):
    from ctypes import *;
    cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/libdmtx.so');
```

```
  def onLoad(self):
            import sys, os, ZBar, Image, sip, string;
    from ctypes import *;
    cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/libdmtx.so');
    cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/CORE_RL_magick_.lib');
            cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) +
'/lib/CORE_RL_Magick++_.lib');
            cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) +
'/lib/CORE_RL_wand_.lib');
            cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/libZBar-
0.def');
            cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/libZBar-
0.lib');
            cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/X11.lib');
```

Ces librairies se trouvaient dans *C:\Program Files (x86)\Aldebaran\Choregraphe 1.12.0.62\lib\ctypes\lib*

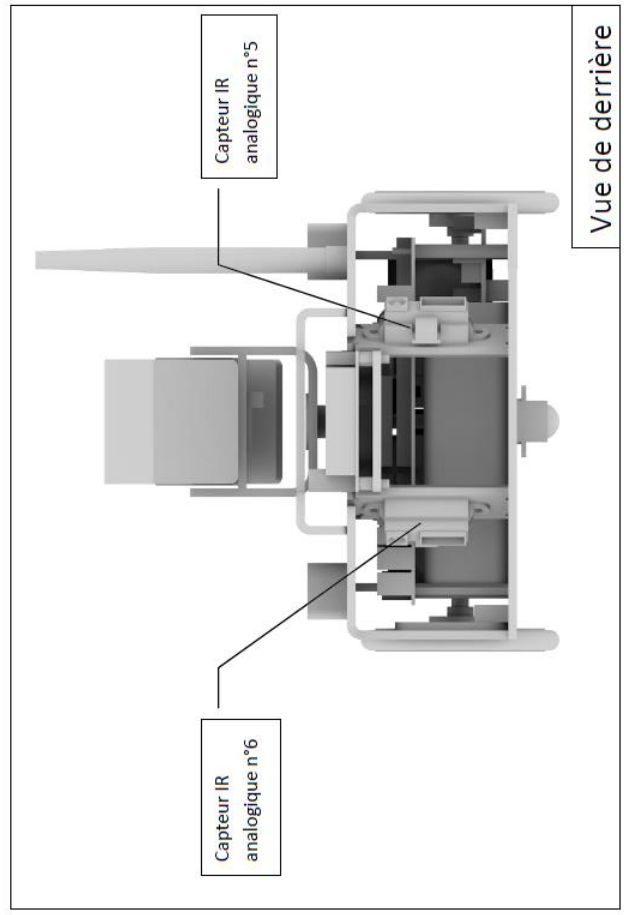
Cependant, les librairies n'étaient pas localisées lors de l'exécution.

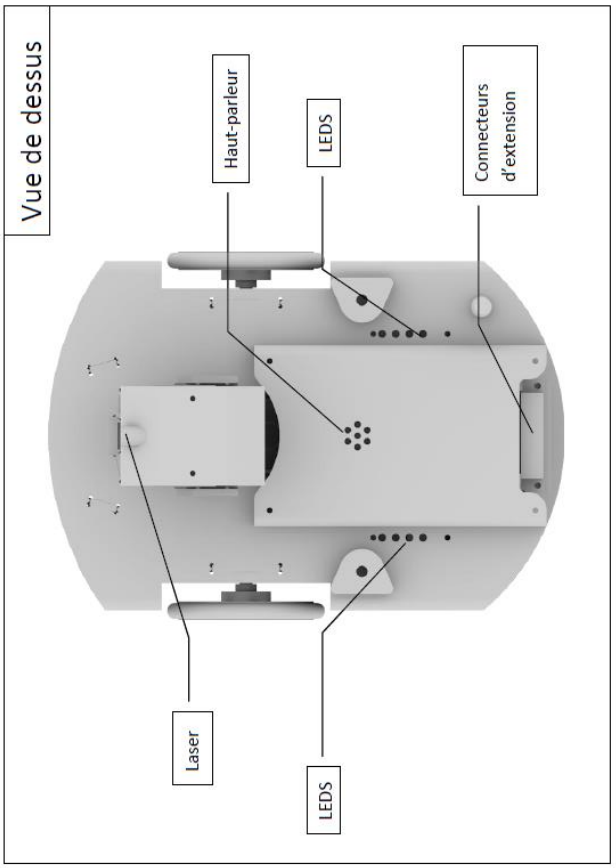
Cet alors, qu'un développeur du projet d'Aldebaran nous indiqua qu'il fallait linker les librairies statiquement. Nous n'avons plus continué sur ce point puisque la nouvelle plateforme Emox venait d'arriver et on arrêta cette partie du projet. Il faudrait donc, à présent linker les librairies nécessaires (ZBar et ImageMagick si on ne la change pas par OpenCV comme nous en avons expliqué précédemment).

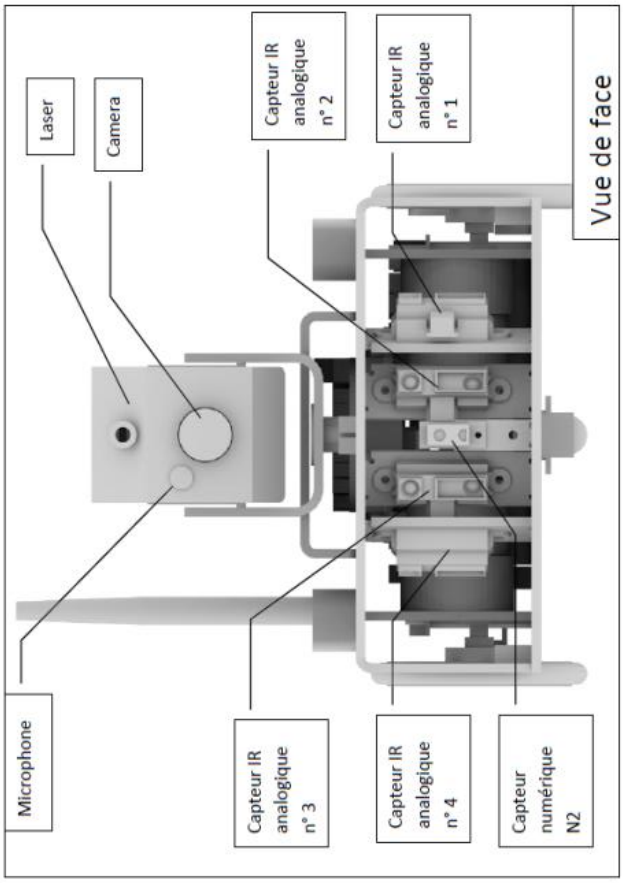
3. EMOX

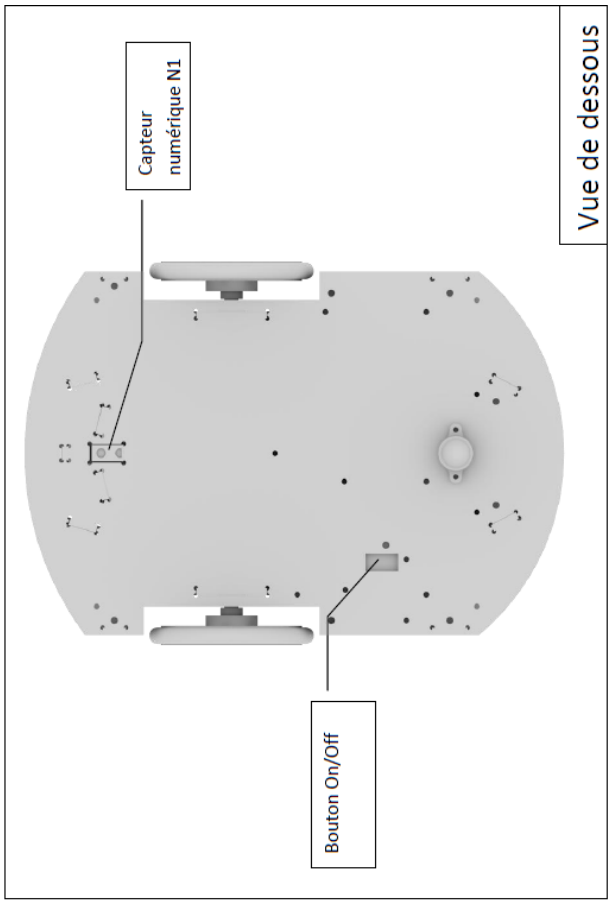
3.1.Description technique

Nous parlerons des détails techniques de cette version prototype d'Emox, l'Emox Lab. En principe, la nouvelle version qui arrivera en mars sera équipée avec presque les mêmes caractéristiques, mais nous ne sommes pas certains pour l'instant. Dans cette section, nous ne nous arrêterons pas dans la description des éléments mécatroniques mais aussi nous parlerons de ses résultats pendant son fonctionnement et les commandes du control dans la console du logiciel.


Vue de derrière
Capteur IR analogique n°5
Capteur IR analogique n°6

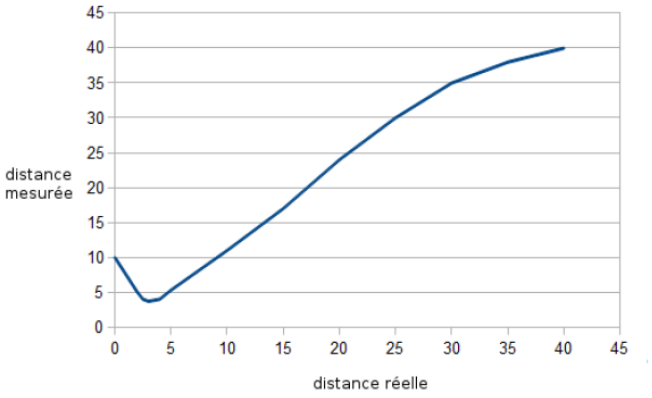

Vue de dessus
Haut-parleur
LEDS
Connecteurs d'extension
Laser
LEDS


Vue de face
Laser
Camera
Capteur IR analogique n° 2
Capteur IR analogique n° 1
Microphone
Capteur IR analogique n° 3
Capteur IR analogique n° 4
Capteur numérique N2


Vue de dessous
Capteur numérique N1
Bouton On/Off

Pour la lecture de l'environnement il est composé de :

- **Six capteurs (Sharp GP2Y0A41SK0F) de distance analogiques infrarouges** qui détectent des obstacles posés entre 4 et 40cm (datasheet: http://www.dema.net/pdf/sharp/GP2Y0A41SK0F.pdf). Il faut remarquer que la mesure minimale est 4 cm car la distance des objets disposés à une distance inferieure peut ne pas correspondre à la celle qu'on a réellement telle que l'image ci-dessous nous montre :



Comparaison entre la distance réelle en centimètre de l'obstacle et la valeur mesurée par le capteur. On remarque qu'à courte distance, c'est-à-dire à moins de 4 cm, la valeur mesurée peut correspondre à deux distances réelles.

Ils sont situés :

- **4 en face** positionnés de telle façon qu'il puisse reconnaître ce qu'il a juste devant et sur les deux côtés.

COMMANDES (valeurs retournées en cm)	
Capteur droit	`distanceR.val ;`
Capteur frontal droit	`distanceFR.val ;`
Capteur frontal gauche	`distanceFL.val ;`
Capteur gauche	`distanceL.val ;`

- **2 à l'arrière** pour la même raison.

COMMANDES (valeurs retournées en cm)	
Capteur de derrière droit	`distanceBR.val ;`
Capteur de derrière gauche	`distanceBL.val ;`

– **Deux capteurs de distance numériques infrarouges** (Sharp GP2Y0, datasheets: http://www.alldatasheet.com/view.jsp?Searchword=GP2Y0) :

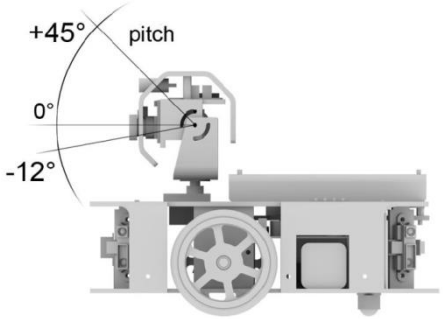
- **Distance < 5cm** (Sharp GP2Y0D805Z0F) situé dans la partie du dessous pour détecter s'il est appuyé sur une surface. Son principale utilisation est pour le programmer de telle façon que s'il est pris et élevé, qu'il arrête le mouvement des roues et qu'il mette en pause le module.
- **Distance < 10cm** (Sharp GP2Y0D810Z0F) devant supportant les capteurs analogiques pour prévenir les collisions.

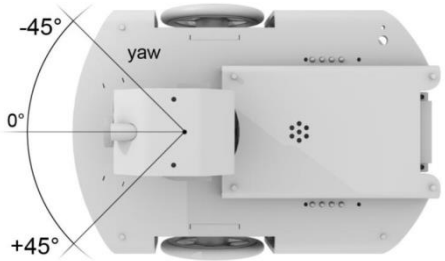
COMMANDES (valeurs retournées : 0 ou 1)	
Capteur frontal	`proximitF.val;`
Capteur dessous	`proximityD.val;`

Dans la partie mécanique il est équipé de deux pairs de moteurs pour le mouvement des rues de la tête:

– **Deux servomoteurs pour la tête** (datasheet : http://www.pololu.com/file/0J507/HD-1581HB.pdf). Afin de savoir sa position, ils sont montés avec un potentiomètre analogique chacun. Les relations entre les valeurs de tension analogiques et les valeurs numériques envoyées sont :

- Axe horizontal (Yaw) :
  0.2V et 4.2 V == 800 et 2200
- Axe vertical (Pitch) :
  2.2V et 3.9V == 800 et 2200





COMMANDES		
Axe vertical	Activation du moteur : 0 - désactiver 1- activer	`headPitch.load=1 ;`
	Connaitre position (retournée en radians)	`headPitch.position;`
	Donner position (tapé en radian -pour degré il faut ajouter *deg* à la fin-)	`headPitch.val = valeur;`
Axe horizontal	Activation du moteur : 0 - désactiver 1- activer	`headYaw.load=1;`
	Connaitre position (retournée en radians)	`headYaw.position;`
	Donner position (tapé en radian -pour degré il faut ajouter *deg* à la fin-)	`headYaw.val = valeur;`

Remarque : Avant de mettre la valeur de l'angle, il faut tout d'abord activer les servomoteurs en entrant les commandes `headYaw.load=1;` et `headPitch.load=1;`

- **Deux moteurs DC/DC de 12V et 0.3A pour les roues,** (datasheet : http://datasheet.octopart.com/HN-GH12-1634T-R-Jameco-Reliapro-datasheet-7274139.pdf) avec une vitesse maximal de 200 tr/min, soit 40 cm/s.

  Afin de connaître sa position, ils sont équipés avec deux disques encodeurs de 13 trous chacun, ce qui donne un total de 390 pas per tour de roue complète. Le capteur chargé de cela est un Sharp GP1A53HR (datasheet : http://pdf1.alldatasheet.com/datasheet-pdf/view/42773/SHARP/GP1A53HR.html).

COMMANDES		
Roue droite	Position (retournée en cm)	`wheelR.position ;`
	Mouvement (tapé en cm/s)	`wheelR.val =` *valeur ;*
Roue gauche	Position (retournée en cm)	`wheelL.position;`
	Mouvement (tapé en cm/s)	`wheelL.val =` *valeur ;*
Toutes les deux roues	Mouvement (tapé en cm/s)	`wheel.val =` *valeur ;*

- **Une bille** au-dessous et à l'arrière comme support physique.

Pour la partie multimédia :

- **Une caméra VGA 30fps de résolution 640 x 480** dans la tête connecté par USB (datasheet : http://www.datasheetdir.com/OV530-B49+download)

COMMANDES	
Obtenir l'image courante	`camera.val;`
Angle d'ouverture verticale (ret. en radians)	`camera.xfov;`
Angle d'ouverture horizontale (ret. en radians)	`camera.yfov;`
Largeur de l'image (ret. en pixels)	`camera.widthCapture;`
Hauteur de l'image (ret. en pixels)	`camera.heightCapture;`

- **Un microphone** dans la tête à côté de la caméra.
- **Un haut-parleur** monté sur la carte.
- **Un laser ligne** rouge sur la caméra.
- **8 LEDs** en deux groupes de 4 (une rouge, deux blanche, une verte) qui informent de l'état des connexions wifi, soit :

  - **LEDs rouges** : si fixes, cela signifie que le robot est allumé.

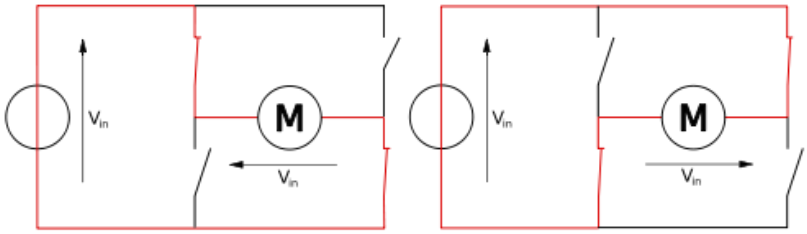
- **LEDs blanches** :
  - Droite (ils montrent l'état de la connexion au réseau):
    - Clignotantes : le recherche le réseau wifi. Ici nous avons deux cas : que ce soit le robot lui-même qui crée le réseau (avec un algorithme à part) ou qu'il y ait une autre créée par un router. Dans tous les deux, le robot se connectera au réseau qu'il a comme favorite (par défaut, son propre réseau ad-hoc).
    - Fixes : le robot est connecté au réseau.

  - Gauche (ils montrent l'état de la connexion à l'Emox Soft):
    - Clignotantes : Le robot n'est pas encore connecté.
    - Fixes : Après d'avoir sélectionné le robot depuis l'onglet de connexion du logiciel, on doit atteindre et finalement ces leds deviendront fixes. Le robot est connecté est prêt pour le lancement du code.

- **LEDs vertes** : pas d'utilisation pour l'instant.

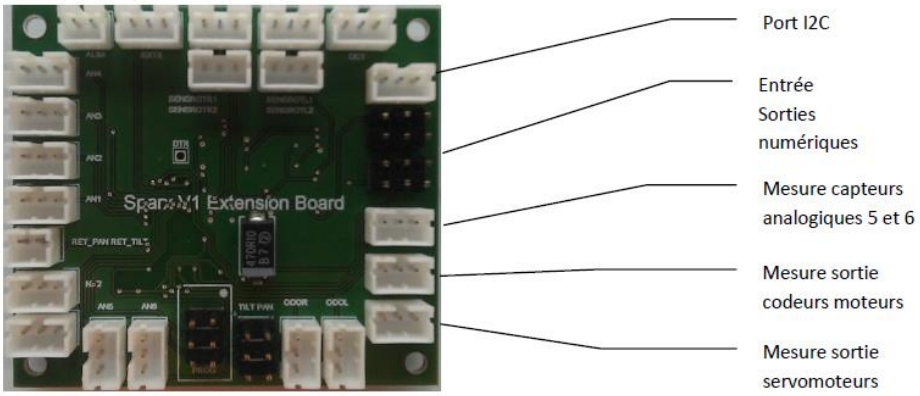
Pour gérer le fonctionnement de tous les capteurs et du système Linux embarqué, le robot est équipé de deux cartes électroniques :

- **La carte mère avec un processeur Arm9 @ 200 MHz** et :
  - **Une puce Wifi** intégrée.
  - **Un pont H** pour la gestion des moteurs. Ce circuit électronique permet aux moteurs de tourner dans les deux sens de rotation. Les schèmes des deux positionnements d'un pont H simple sont :



  - **Un système Linux** embarqué.

- **Une carte Arduino d'extension avec un microcontrôleur Atmel @ 8Mhz** (http://www.arduino.cc/fr/). L'avantage de cette carte c'est qu'on dispose des ports libres en plus pour y intégrer des autres capteurs ou moteurs.



Port	Photo	Pin 1	Pin 2	Pin 3
I2C (non fonctionnel)		Data	Clock	GND
Entrée sortie numériques Arduino (numéro de port arduino en rouge)		Signal	VCC	GND
Capteurs 5&6		Valeur capteur 6	GND	Valeur capteur 5
Sortie codeur moteur		Codeur Droit	GND	Codeur Gauche
Signaux PWM servomoteurs		Signal PWM2 (PAN)	GND	Signal PWM1 (TILT)

3.2. Installation et connexion

Après l'installation de la première version du logiciel (***Emox Soft 1.0.0***) nous n'avons pas réussi à nous connecter correctement au robot et nous n'avons pu démarrer aucun module. Nous avons suivi le Manuel d'utilisation et contacté avec M. Faguet pour chercher une solution.

Dans un second terme, M. Faguet, nous a fourni une deuxième version du logiciel, ***Emox Soft 1.0.1*** avec lequel nous avons finalement réussi à nous connecter au robot.

Ce logiciel permet de travailler en simulation, avec un environnement virtuel ou bien en mode connexion avec le robot réel. Pour travailler avec le robot virtuel, il suffir de lancer l'Emox Soft et le robot virtuel et par défaut choisit.

Pour se connecter au robot réel, il faut suivre certains pas avant le lancement du logiciel. Quelques-uns ne doivent pas être oubliés et rajoutés aux indications trouvées dans le tutoriel d'Awabot:

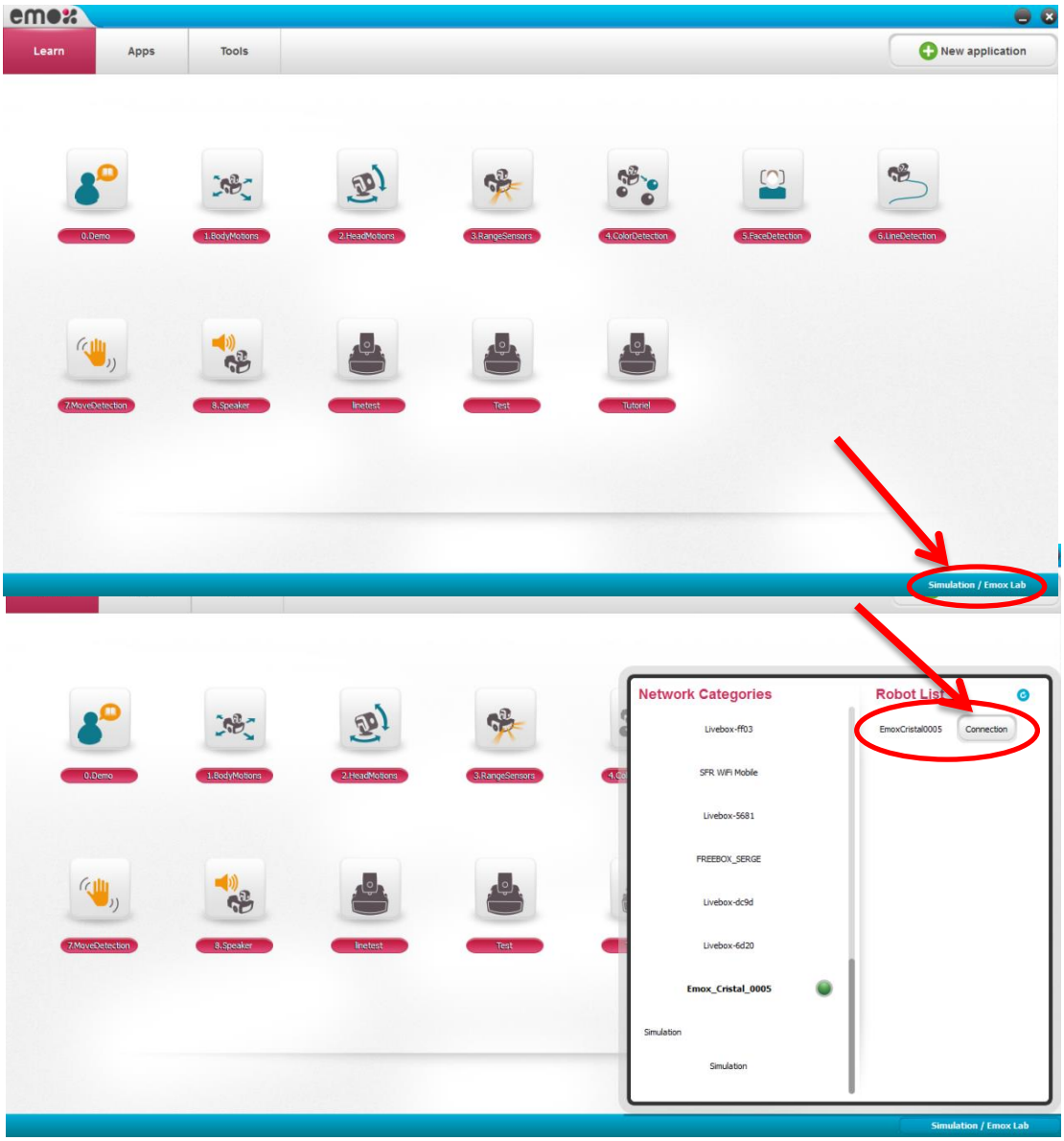
1) Allumez le robot et testez que les leds blancs droits soient fixes et ceux de la gauche clignotants.

2) Cherchez le réseau ad-hoc créé par Emox (dans notre cas, elle est appelée Emox_Cristal_0005) et se connecter.

3) Pendant toute l'utilisation du logiciel, il faut que le firewall de Windows et éventuellement s'il existe, celui de l'antivirus, soient désactivés. Une autre solution possible est de créer des exceptions dans tous les deux cas pour le logiciel (si l'adresse du logiciel n'est pas changée pendant l'installation, normalement on devra mettre : C:\Emox Soft\system\bin\EmoxSoft.exe).

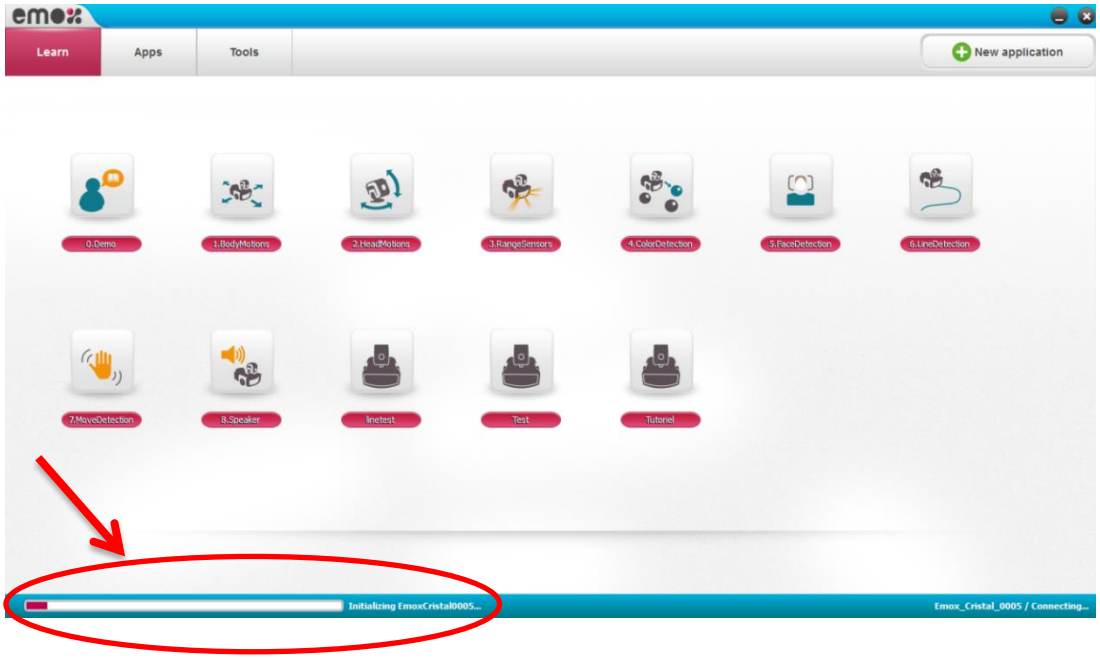
4) Lancez l'Emox Soft en tant qu'administrateur pour éviter de futurs problèmes pendant l'utilisation d'un module.

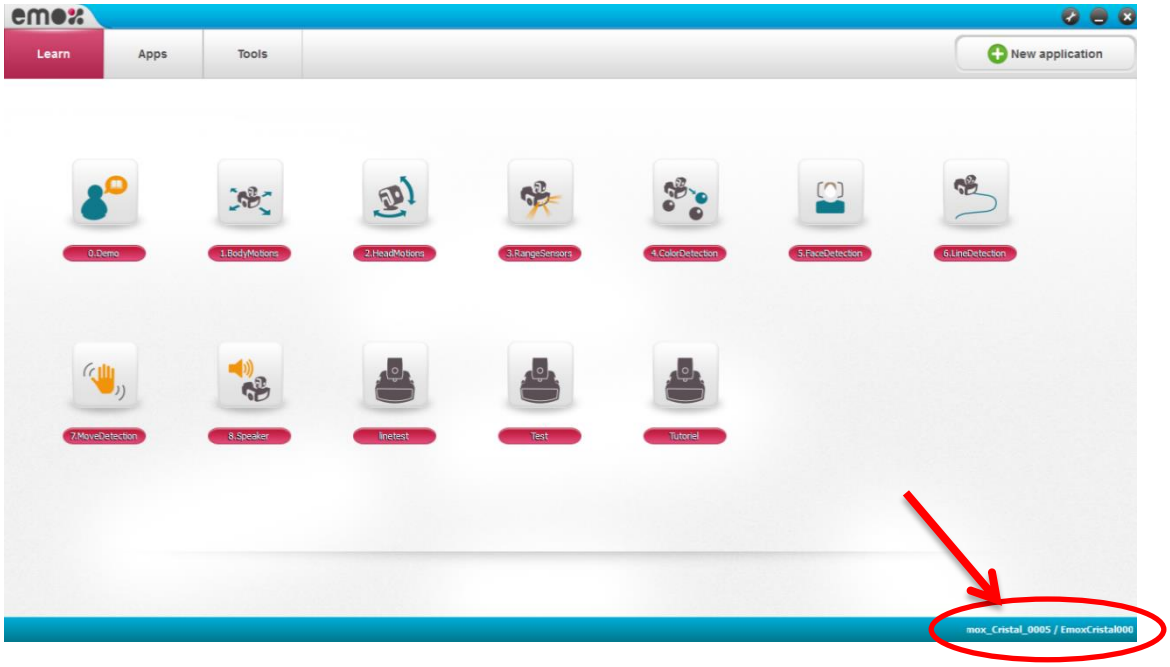
5) Dans le logiciel, cliquez sur le bouton « Simulation/Emox Lab » situé en bas à droite sur la barre bleue. On devra trouver un onglet comme celle de l'image suivante :



La première fois qu'on démarre le logiciel après d'avoir allumé le robot, ce n'est pas rare qu'on ne trouve pas le robot dans la liste située à droit (le robot doit prendre son temps pour être prêt) Si c'est le cas, il ne faut que cliquer sur le bouton bleu Reload ou bien si cela ne suffit pas, relancer l'Emox Soft.

6) Attendez que la barre rose de connexion sur la barre bleue soit complète et afin de tester qu'on est bien connecté et que on n'a pas des autres soucis, démarrez une application (si c'est votre première connexion voir 7). Si tout est bien passé, on devrait obtenir un écran comme celui-ci-dessous :





Une fois connectés, si on trouve que le logiciel n'arrive pas à montrer aucune image de la caméra d'Emox ou qu'il se plante, cela peut être causé par :

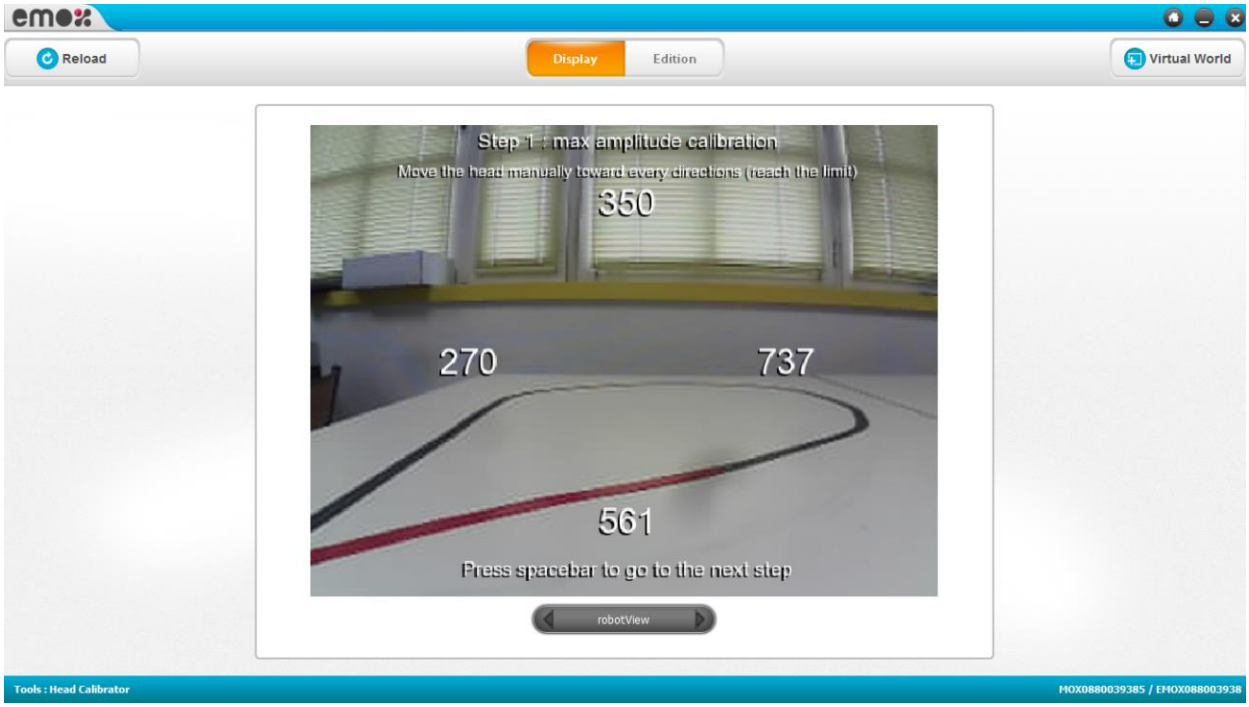
- Le firewall de l'antivirus n'est pas bien désactivé : Nous avons apprécié que quand on a oublié désactiver celui de l'antivirus, quand on démarre une application (par exemple, « Head Calibrator ») le logiciel ne montre aucune image, ne répond pas, puis plante finalement.

- Problème associé à la carte vidéo : Il est dit dans le Manuel d'utilisation qu'une requise d'hardware est d'avoir une carte vidéo avec 512 Mo de Ram dédiée.

7) En cas de première connexion avec la plateforme, il faut lancer avant tout une application qui permet de calibrer les moteurs de la tête. Cette étape permet de protéger les servomoteurs. Elle est appelé « Head Calibrator » et on la trouve dans l'onglet « Tools ».

Quand on démarre l'application on devra voir l'image captée par la caméra du robot et quatre numéros situés à droit, à gauche, en bas et en haut. Ces numéros indiquent les valeurs de rotation des servomoteurs. Suivez les indications montrées : bougez la tête jusqu'au maximum selon les deux axes de rotations.

Attention : bougez la tête doucement et ne forcez pas les limites du mouvement.
Après suivre les indications, les valeurs que nous avons obtenues sont :



Il faut remarquer que ces valeurs ne doivent pas être forcément celles que vous obtiendrez.

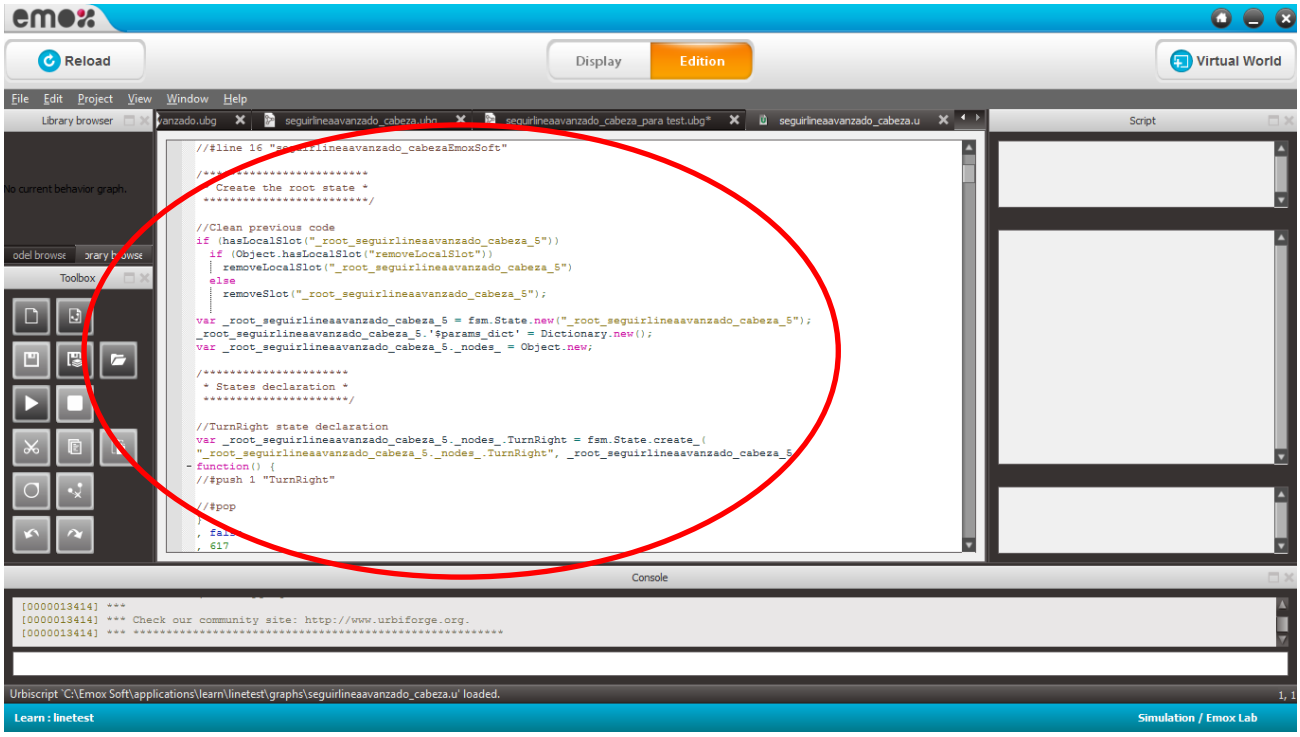
3.3.Programmation du robot

Emox est une plateforme qui fonctionne avec Linux, et entièrement programmable grâce au langage Urbi. Celui-ci se compose des UObject du C++ et du langage événementiel urbiscript. Urbi est né à l'ENSTA Paristech en 1999 puis est devenue partie de Gostai en 2006. C'est cette partie qui rend possible la programmation graphique.

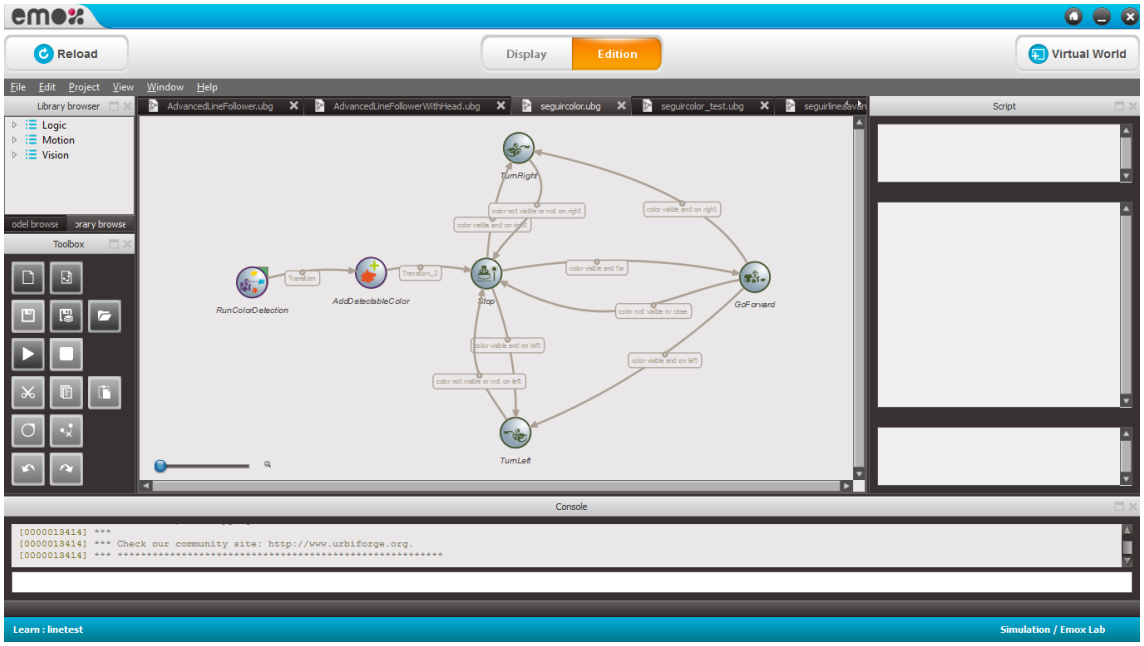
Pour programmer Emox, on utilise l'Emox Soft. Ce logiciel permet de se connecter au robot et gère la connexion PC-robot via wifi. Il peut jouer le rôle de lanceur d'applications et d'éditeur. Nous pouvons donc programmer directement nos modules avec le logiciel puis les exécuter. Les applications sont exécutées sur l'ordinateur. De cette façon, le robot renvoie les valeurs des capteurs à l'ordinateur, qui prendra les décisions et les fera réaliser par Emox.

Une fois que l'on a lancé le programme, nous pouvons donc choisir entre trois options de programmation :

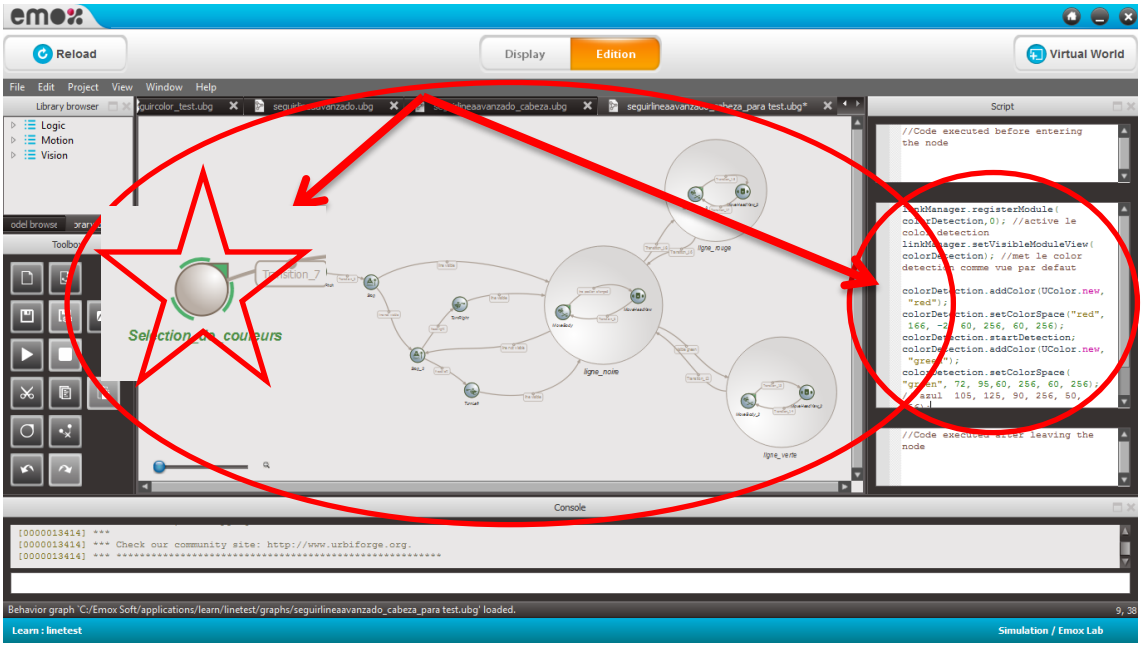
- Programmation en script (SDK)



- Programmation graphique



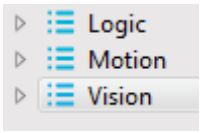
- Programmation mixte



La première méthode est complexe, parce qu'il faut connaître le langage Urbi et ce n'est pas si facile même si on connaît le C++. Et aussi car pour l'instant, la documentation du SDK et de l'API n'est pas complète. Après quelques contacts avec des employés d'Awabot via téléphone ou forum, on peut confirmer la difficulté à dominer l'API puisqu'on a reçu certaines réponses contradictoires.

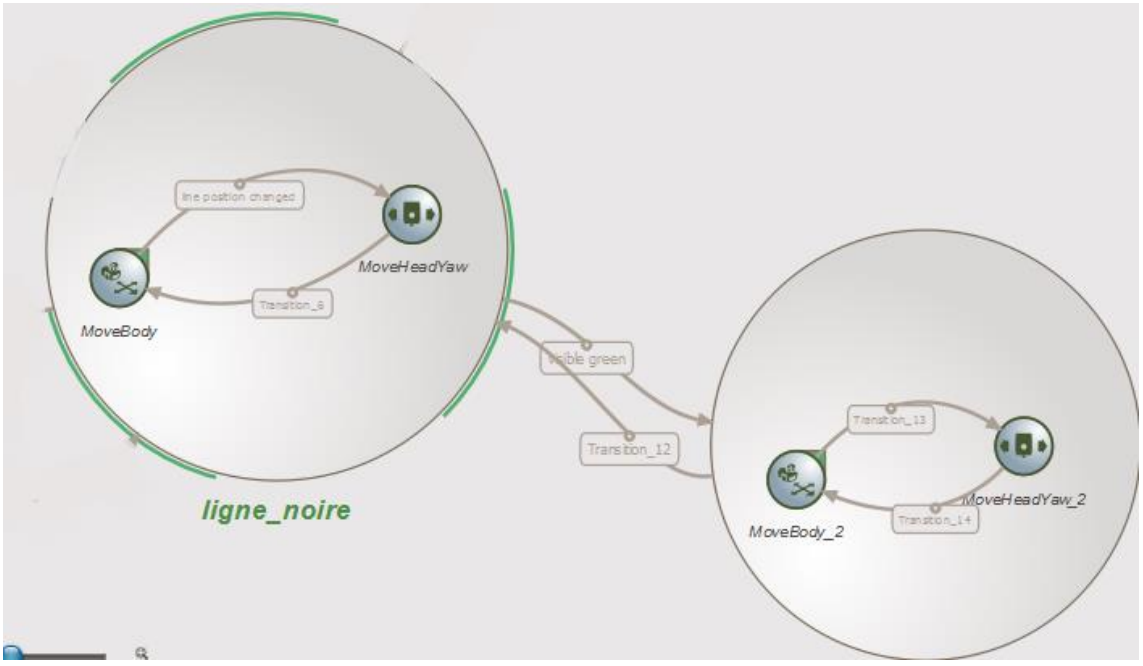
Ensuite la méthode graphique est plutôt simple à utiliser mais ne permet pas le développement d'applications trop complexes. Elle ressemble à la

programmation utilisée dans des champs tels que l'automatique (langage graphcet par exemple des automates). Dans le cas d'Emox, elle se sert de 3 bibliothèques principales



À l'intérieur, on retrouve les nœuds que nous utiliserons lors de la programmation.

Ensuite ces nœuds, qui peuvent être ralliés, sont reliés par des transitions que l'on peut programmer. Ceci est assez utile et offre une grande versatilité.



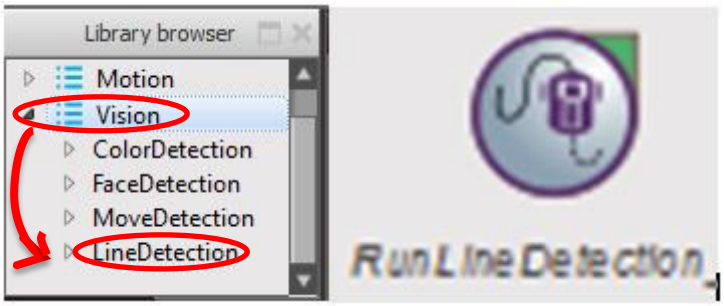
Finalement, une combinaison entre l'interface graphique et l'API est une méthode performante pour programmer l'Emox. En effet, l'interface graphique permet d'exploiter la programmation évènementielle d'une façon plus intuitive et la partie en script nous permettra de créer nos propres nœuds. Ceux-là pourront être plus complexes que les existants ou bien tout simplement, ils seront mieux adaptés à notre application.

Pour apprendre comment écrire le code de façon graphique, nous avions utilisé le tutoriel fourni par Awabot [7]

3.4.Suivi d'une ligne

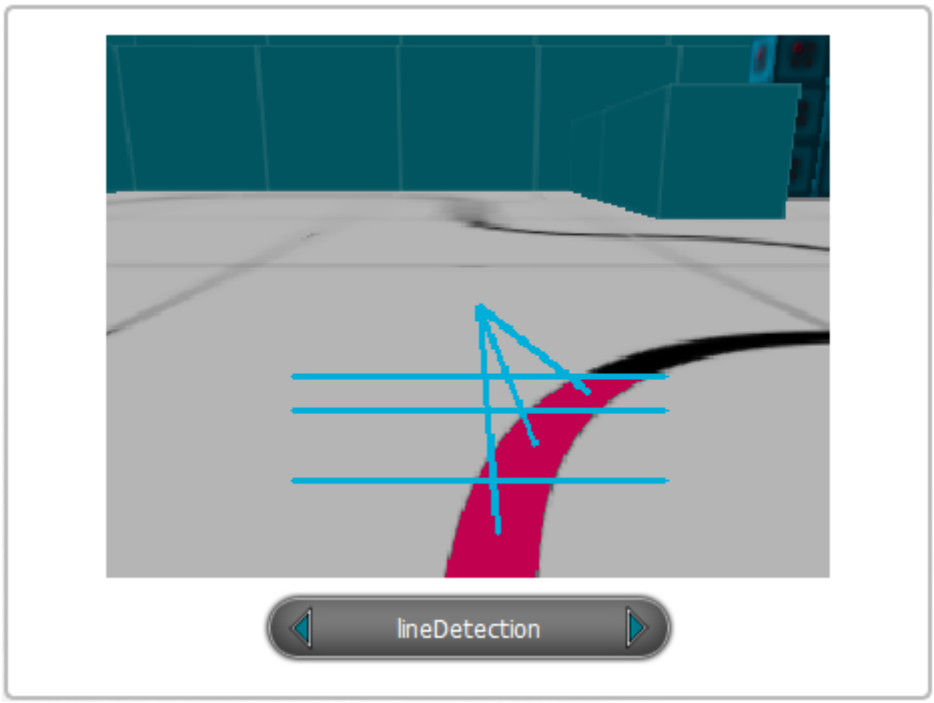
Comme nous avons expliqué dans la présentation du projet, nous voulons qu'Emox soit capable de suivre une ligne sur le sol, tout en variant son comportement en fonction de la couleur de celle-ci. Pour cela, nous avons séparé le problème en deux sous parties : le suivi de la ligne et la détection de couleurs.

Le suivi de la ligne est une façon de montrer le chemin à suivre pour le robot. C'est une partie sur laquelle Awabot a travaillé, en créant un module qui facilite le suivi d'une ligne. Celui-ci se trouve dans la bibliothèque Vision.



Grâce à Line Detection, la moitié inférieure de l'image captée par la caméra est divisé en 3 régions qui permettent d'évaluer la position de la ligne à courte, moyenne puis longue portée. On peut alors connaître la position du centre de gravité de la ligne dans cette région avec la commande :

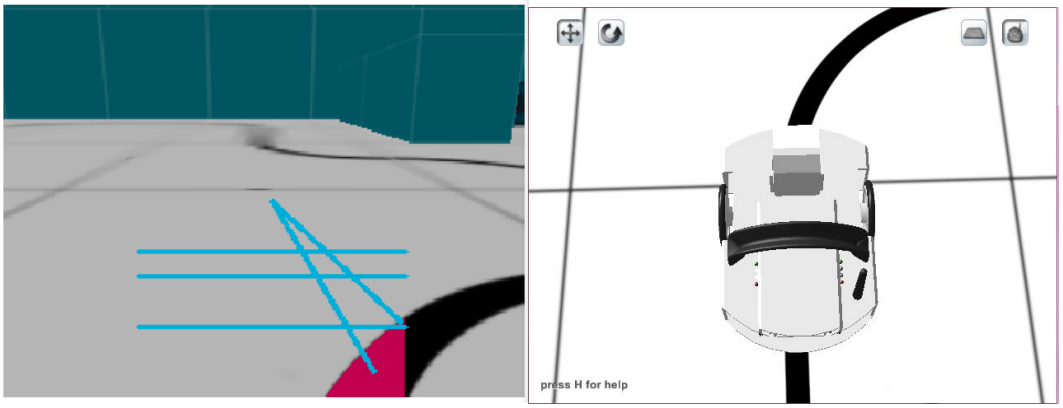
lineDetection.getZone(zone).x



Une fois connue la position de la ligne, le robot doit être capable de la suivre. Pour cela, il y a plusieurs façons mais la plus intéressante est que le robot suive la ligne avec la tête, puis aligne le corps pour s'orienter dans la direction de la ligne. Ainsi c'est la tête qui dirige, ce qui permet de ne pas perdre la ligne. Ces actions d'alignement du corps et orientation de la tête se succèdent très rapidement, de sorte qu'il n'y ait pas de pause et que tout semble simultané.

Nous allons l'illustrer avec un exemple :

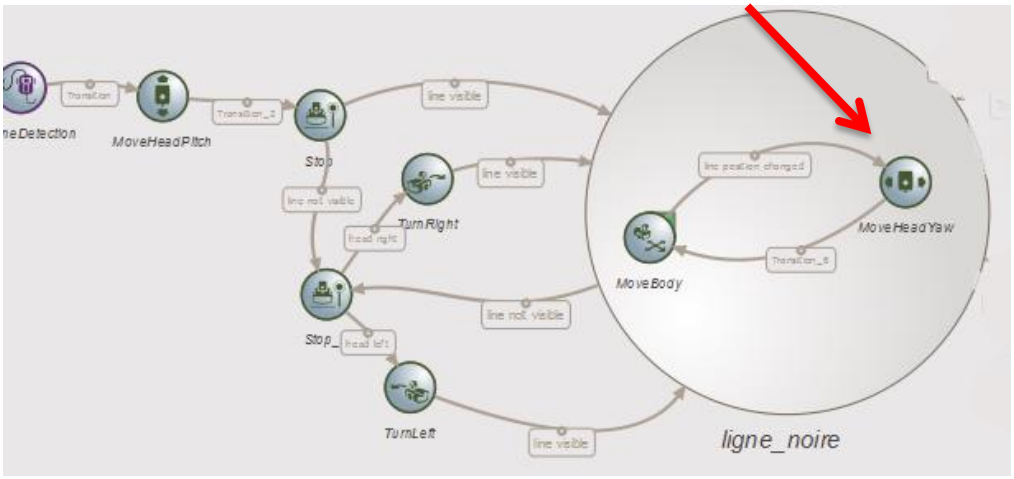
- Nous observons que l'on arrive à un virage, les positions x de la ligne se déplacent dans la direction du virage :



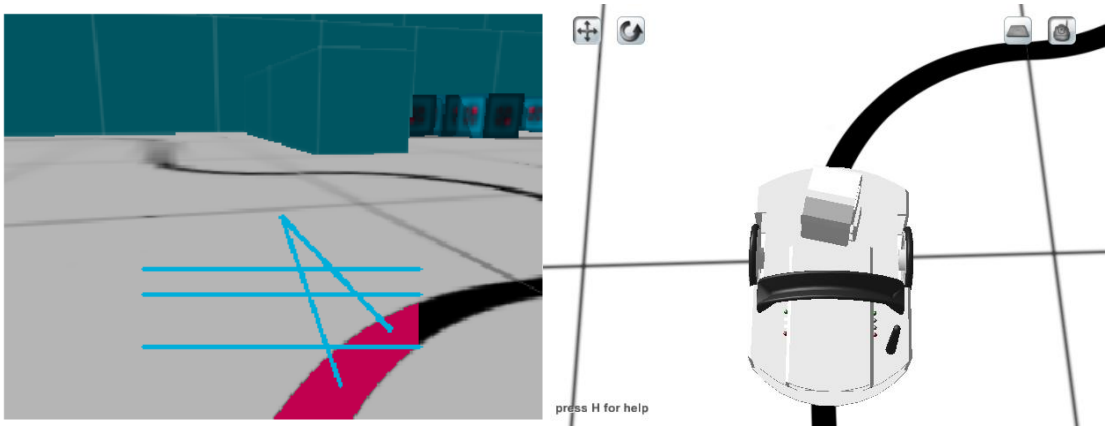
- À ce moment, la tête s'oriente pour s'aligner avec la ligne. Pour cela nous utilisons :

```
headYaw.val = headYaw.val - (0.1 *
                lineDetection.getZone(0).x + 0.15 *
                lineDetection.getZone(1).x + 0.2 *
                lineDetection.getZone(2).x)
```

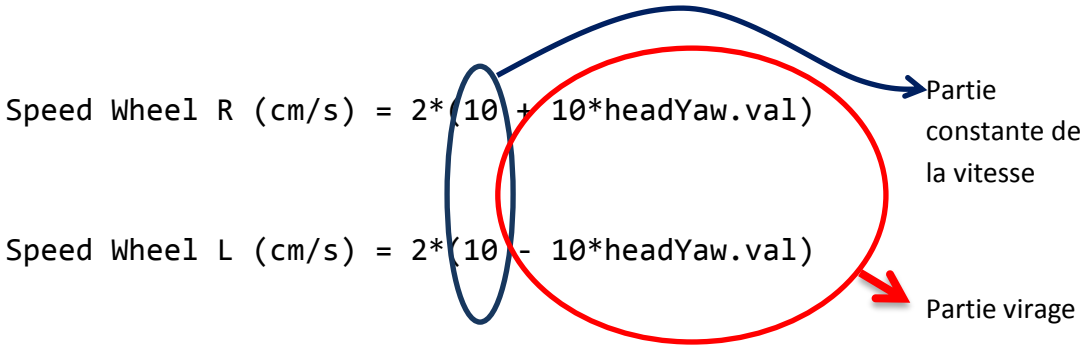
Ce code s'introduit dans les paramètres du noeud :



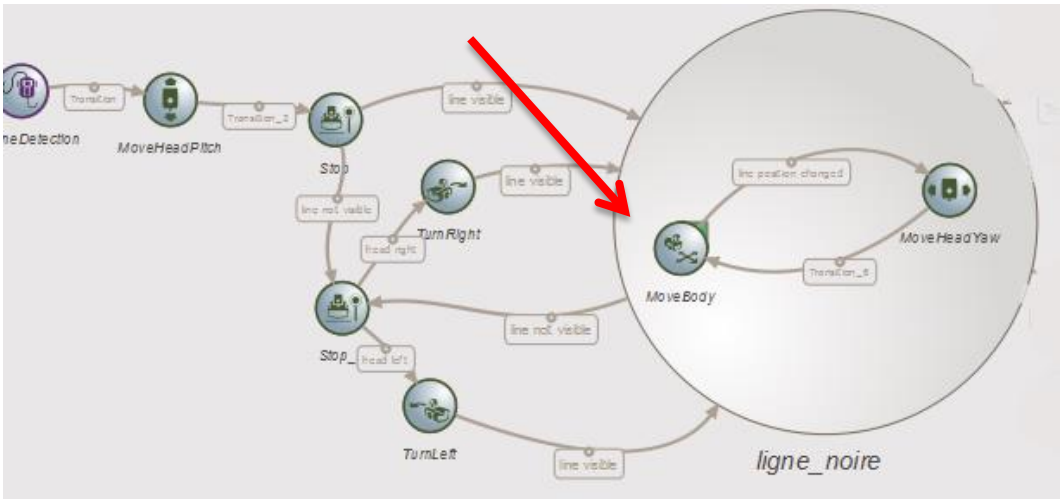
Comme la tête tourne positivement vers la gauche, tandis que les valeurs de la ligne selon l'axe x sont positives vers la droite, nous devons mettre un signe négatif. Ensuite nous appliquons des coefficients aux valeurs des différentes zones. Si les paramètres sont trop élevés, les corrections de position de la tête sont trop importants et le robot part d'un côté puis d'un autre sans précision. Nous avons choisis ces valeurs en essayant plusieurs combinaisons. Nous avons testé nos valeurs sur un code d'Awabot qui n'était pas capable de suivre la ligne et avec nos valeurs, le robot est capable de la suivre, ce qui prouve l'amélioration du système.



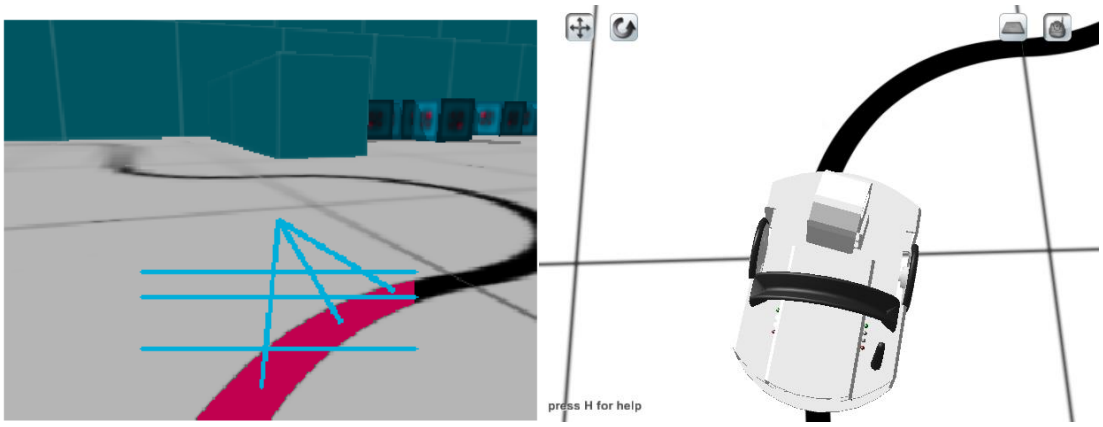
- Finalement, tout en continuant d'avancer, le corps s'aligne avec la position de la tête et donc, avec la ligne.



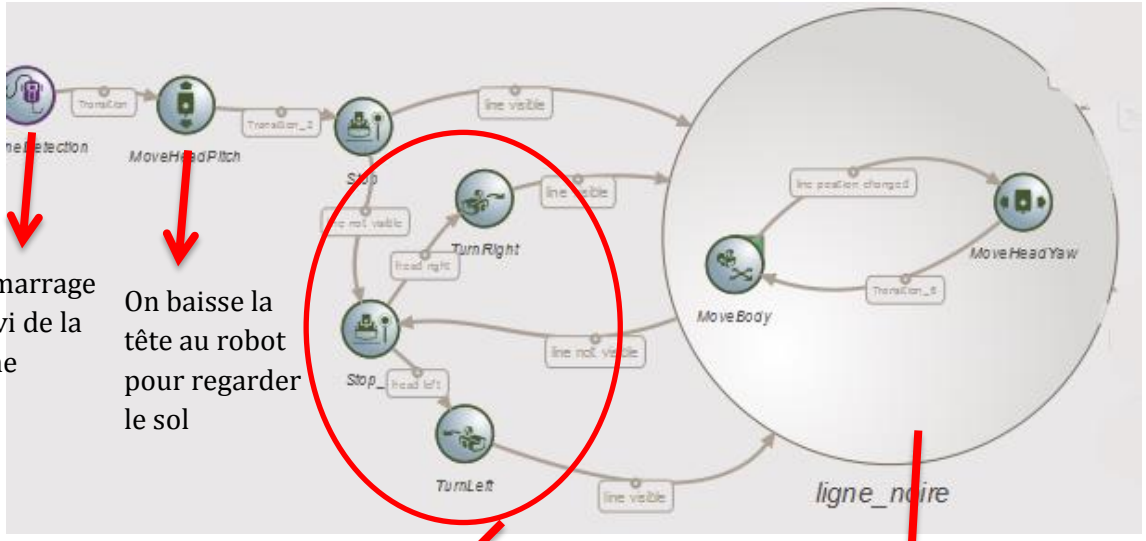
Emplacement de ces paramètres :



Le robot se déplace donc à une vitesse constante de 20 cm/s et tourne avec une valeur 20 * (orientation de la tête) cm/s. Comme il y a une partie constante dans la vitesse, le robot va avancer de façon constante et les corrections de position pour suivre la ligne vont être intégrées dans le mouvement, de sorte que l'utilisateur le perçoit comme une seule action.



Le graphe est le suivant :



Démarrage suivi de la ligne

On baisse la tête au robot pour regarder le sol

Si le robot perd de vue la ligne, il tourne sur lui-même pour la retrouver

Nœud qui gère l'alternance entre le mouvement de la tête pour regarder la ligne puis du corps pour s'aligner

3.4.1. Possible amélioration du hardware

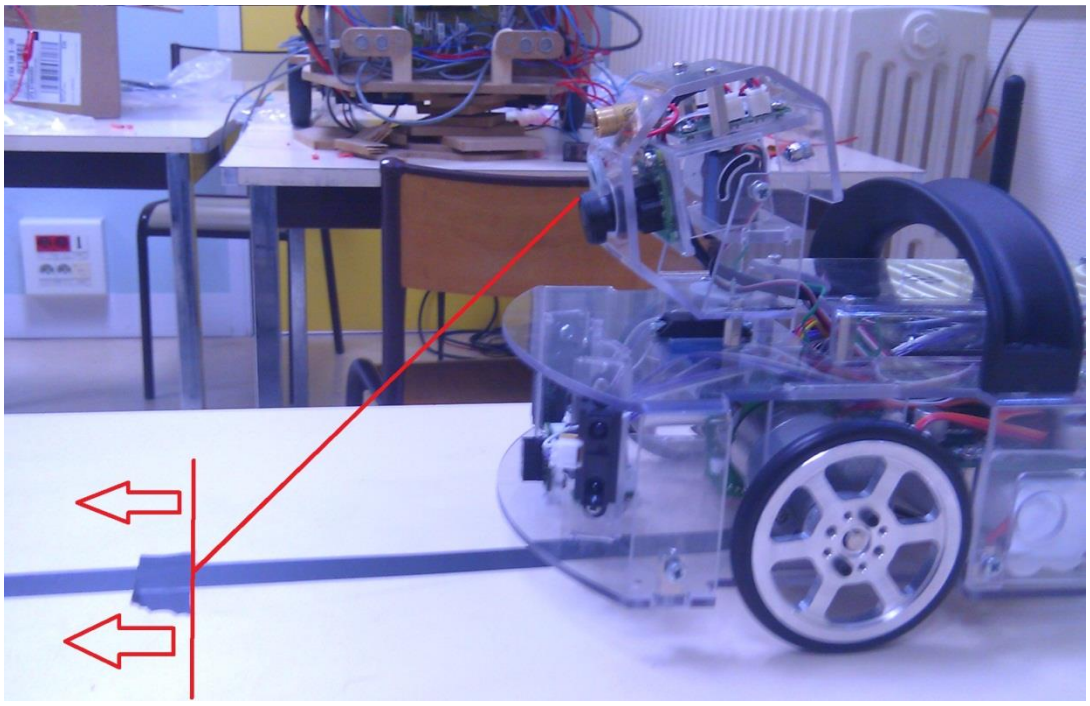
Il existe un projet dans le laboratoire de mécatronique de l'ENSAM avec un robot de la marque de jouets Lego, connu comme Lego Mindstorms NXT 2.0.

Ce projet est très proche au notre car ils ont construit une robot-voiture qui doit suivre une ligne et réaliser différents mouvements selon les couleurs. Par rapport à Emox, le robot de Lego emporte un capteur très utile pour cette fonction, un capteur de couleur. C'est le capteur RGB montré dans l'image ci-dessus avec lequel on peut reconnaître le couleur, l'intensité de lumière et, en plus, l'utiliser en tant qu'un lampe:

http://botbench.com/blog/2011/09/16/exposed-lego-colour-sensor/

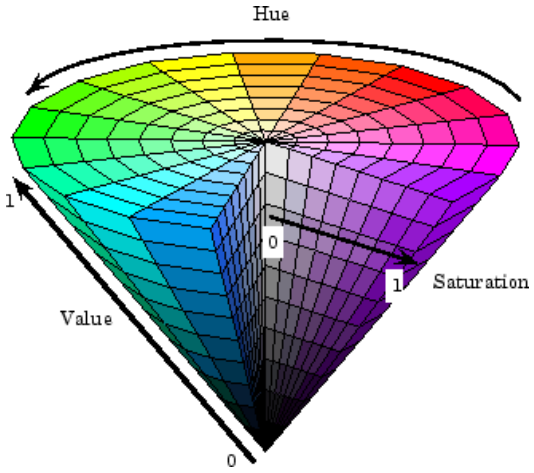
Colour Sensor
Detects different colours and light intensity.
Use it as Colour Sensor, Light Sensor or as a lamp!
Colour Sensor!
Light Sensor!
Lamp!

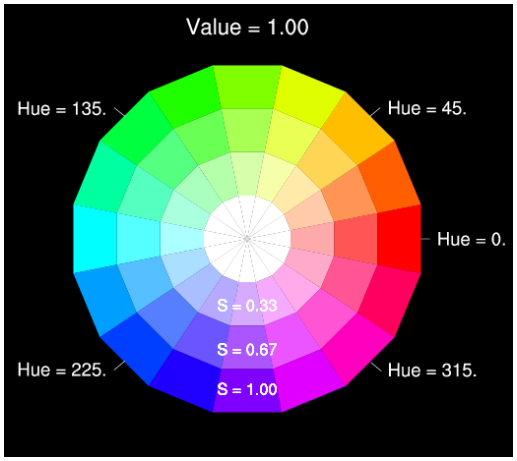
Nous avons proposé à M. Faguet d'équiper à Emox avec un capteur RGB. Pour l'instant on distingue les couleurs à travers des images captées et lues par le logiciel. Étant donné que le mouvement selon l'axe vertical est très restreint en direction au sol, il y a une différence de 16cm environ entre la ligne qui est sous Emox et le limite de la vision, ce qui fait très faible la performance du robot dans le cas de distinguer les couleurs. L'image présentée ci-dessous montre la limite de vision d'Emox :



3.5.Détection de la couleur

Pour la détection de la couleur, l'échelle que les développeurs ont choisi est le modèle HSV (*Hue, Saturation, Value*, soit en français : Teinte, Saturation, Valeur). Normalement ce modelé est mesuré dans ces écarts:





- Pour la Teinte : 0º - 360º
- Pour la Saturation : 0 – 100
- Pour la Valeur : 0- 100

La différence par rapport aux échelles habituelles est que pour la programmation d'Emox les trois composantes sont mesurées de 0 à 255. Il existe aussi le modèle RGB (Red, Green, Blue ou Rouge, Vert, Bleu en français) qui est très utilisé. Selon ils expliquent dans le tutoriel du robot, ils ont choisi le modèle HSV parce que la teinte est une composante.

Grâce à la bibliothèque de vision du robot, on est capable de reconnaitre un ou plusieurs couleurs posés devant lui et de faire ce qu'on veut avec eux comme les suivre, par exemple. Pour qu'Emox garde les couleurs voulus, nous avons deux façons de les ajouter, ou bien par sélection graphique à travers de l'écran de vision du robot ou bien on peut ajouter manuellement par script les valeurs HSV. Nous choisissons les définir par script car nous voulons développer un module qu'on soit capable de le démarrer de façon autonome sans besoin de sélectionner les mêmes couleurs chaque fois.

Nous avons eu des problèmes pour la définition des couleurs par script dans la programmation graphique, mais grâce au forum dont nous avons parlé avec un autre utilisateur d'Emox Lab qui nous a aidés avec cette question. On peut trouver la question et sa réponse ce dans cette adresse :

http://awabot-redmine.herokuapp.com/boards/5/topics/4

Ce que nous devons écrire est :

```
linkManager.registerModule(colorDetection,0); // active le color detection
linkManager.setVisibleModuleView(colorDetection); //met le color detection comme vue
                                                  par defaut
colorDetection.addColor(UColor.new, "red1");//cree la variable red1
colorDetection.setColorSpace("red1", 137, 153, 90, 256, 54, 256); //couleur violette d'une
                                                  balle du monde virtual
colorDetection.startDetection;
```

Donc à ce moment il ne manque que de connaître les couleurs avec lesquelles on veut travailler.

3.5.1. Comment obtenir les valeurs des couleurs

3.5.1.1. À travers d'une application externe

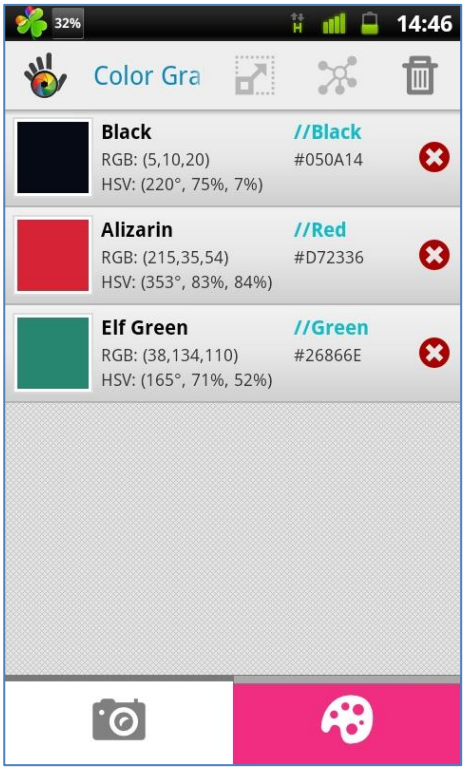
En principe, les informations que nous avions sur le robot étaient que malheureusement il n'existait pas encore la possibilité de savoir les valeurs gardés par Emox à travers de la sélection graphique de l'aire colorée. Après de faire une recherche afin d'avoir une façon simple, intuitive mais surtout qu'elle soit accessible aux futurs élèves du projet, finalement nous avons décidé utiliser une application mobile pour le Smartphone. Étant donné que nous avons tous les deux des terminaux Androïd, nous avons cherché une application dans le Google Play et pour l'instant il y en a deux pour savoir les valeurs HSV des couleurs qu'on pose devant la caméra, soit :

- ***Color Detector*** :
  (play.google.com/store/apps/details?id=com.mobialia.colordetector)

- ***Color Grab*** :
  (play.google.com/store/apps/details?id=com.loomatix.colorgrab)

Après d'avoir testé toutes les deux, nous avons choisi la dernière, *Color Grab,* puisque elle est capable de nous donner les valeurs qu'on voit à travers de la caméra de façon continue et quand on veut garder les valeurs, on fait une photo et elle garde les valeurs dans une liste. Avec *Color Detector* on ne peut que faire des photos pour connaitre les valeurs.

Grâce à l'application mobile *Color Grab* nous trouvons que les valeurs HSV universelles des rubans adhésifs sont celles qui apparaissent dans l'image ci-dessous :



Étant donné que l'Emox Soft ne reconnaît que les valeurs compris entre 0 et 255, nous avons créé une table Excel afin de faire la conversation rapidement. En plus, nous avons créé deux colonnes finales pour calculer directement les valeurs maximale et minimale de chaque composant. Pour la teinte, nous conseillons que

l'écart des valeurs ne soit pas assez grand puisque on peut prendre plusieurs couleurs différentes au même temps. Par rapport à la saturation et la valeur, normalement on fera à l'inverse car on veut que la détection ne soit pas affectée par la lumière.

Donc nous avons mis des écarts de 10 pour les valeurs de la teinte. Nous faisons aussi le calcule de S et V mais nous utiliserons les valeurs fixes (50, 256).

HSV CONVERTISSEUR

	Echelle	0 - 359 | 0- 100		0 - 255	Min	Max
NOIR	H	220	→	156	146	166
	S	75		194	184	204
	V	7		19	9	29

	Echelle	0 - 359 | 0- 100		0 - 255	Min	Max
ROUGE	H	353	→	251	241	261
	S	83		214	204	224
	V	84		217	207	227

	Echelle	0 - 359 | 0- 100		0 - 255	Min	Max
VERT	H	165	→	117	107	127
	S	71		183	173	193
	V	52		135	125	145

Quand nous avons mis ces valeurs dans le script, nous avons constaté que les couleurs pour l'Emox Soft ne coïncident pas avec les couleurs réelles sur l'échelle 0 - 255 que nous croyions. Pour cette raison, nous avons décidé de le faire de la façon suivante.

3.5.1.2. À travers de la console de debug d'Emox Soft

Dans un deuxième temps nous avons mis les valeurs obtenues dans le code mais nous avons trouvé que les couleurs détectées ne correspondaient pas aux trois couleurs souhaitées. Pour cette raison, nous avons mis une question concernant ce soucis dans le forum et nous avons obtenu une réponse très utile : il est possible de savoir les valeurs captées par le logiciel quand on démarre l'application de choix de la couleur par la souris.

Ce qu'on doit faire est lancer l'Emox Soft avec la console de debug. Pour cela et pour connaître les couleurs, il suffit de:

1) Ouvrir une invite de commandes.
2) Se placer dans le dossier contenant l'exécutable emoxsoft.exe.
3) Taper : `EmoxSoft.exe -dev.`
   L'Emox Soft sera lancé avec une console qui nous dira l'information qu'on veut.
4) Chercher le robot et nous connecter comme on fait normalement.

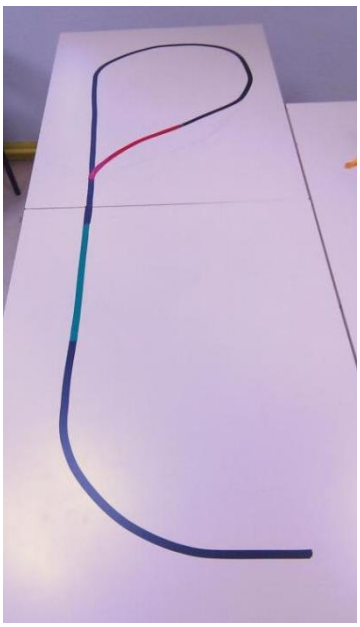
5) Lancer l'application *Color Detection* et sélectionner la couleur souhaitée grâce à la souris. Ensuite on va sur la console de debug et on cherche la dernière ligne qui nous montre la plage de la couleur sélectionnée sous la forme :

```
Color changed to detect H:85-101, S:90-256, V:90-256
```

Les valeurs trouvées grâce à cette démarche sont celles que nous avons mises dans le code pour sélectionner le comportement selon la couleur de la ligne.

3.6.Suivi d'une ligne avec code couleur

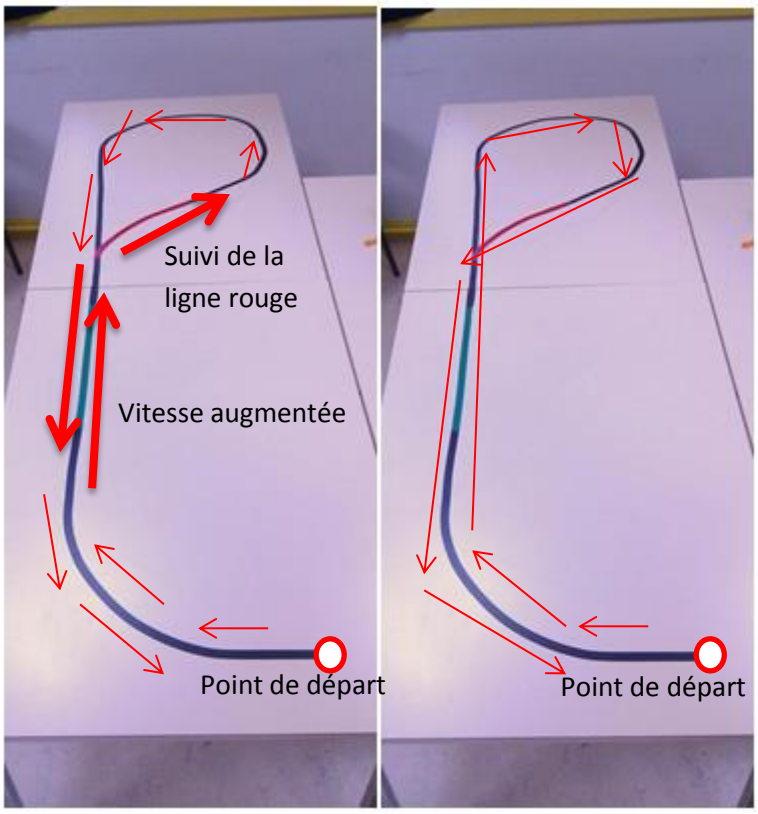
Avec le module de suivi d'une ligne que nous avons amélioré et la détection de la couleur en définissant les couleurs manuellement, nous voulons faire qu'Emox suive un circuit avec de différents comportements en fonction de la couleur de la ligne. Il faut remarquer que dans le module d'Awabot, aucune distinction de couleur n'est faite, et que toutes les couleurs apparaissent comme noir. Nous avons donc, acheté trois rubans adhésifs de couleurs différentes (noir, vert et rouge) pour simuler un circuit :



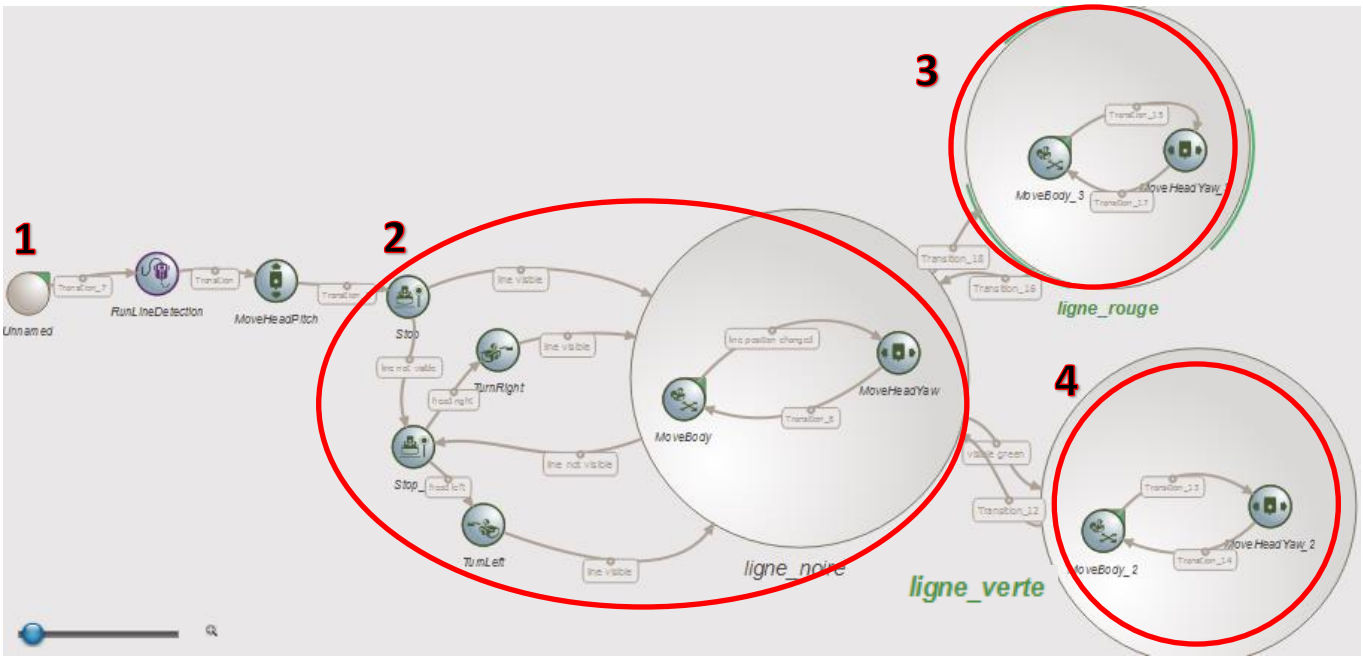
Nous souhaitons que le robot soit capable de suivre la ligne noire tout en obéissant aux comportements préparés par les lignes de couleurs, tout en les suivant. Nous avons décidé qu'une ligne verte indiquera une augmentation de la vitesse et une ligne rouge dans une bifurcation donnera de la priorité à se chemin et se sera donc celui-là qu'Emox devra suivre.

Le comportement sur le circuit serait donc le suivant :

Parcours avec comportement selon la couleur	Parcours sans comportement selon la couleur



Nous avons vue antérieurement comment programmer le suivi d'une ligne et comment détecter les couleurs par script, c'est grâce à ces deux concepts qu'on est capable de concevoir notre module.



1) Lors du démarrage du module, nous introduisant un nœud que l'on programme uniquement avec du script pour sélectionner les couleurs comme expliqué auparavant. Il est cependant, important de signaler qu'il est nécessaire de lancer `colorDetection.startDetection` après introduire une nouvelle couleur:

```
colorDetection.addColor(UColor.new, "red");

colorDetection.setColorSpace("red", 166, -2, 60, 256,
60, 256);

colorDetection.startDetection;

colorDetection.addColor(UColor.new, "green");

colorDetection.setColorSpace("green", 72, 95,60, 256,
60, 256);

colorDetection.startDetection;
```

2) Cette partie est en charge du suivi des lignes noires.

3) Ce nœud gère le suivie de la ligne rouge. Au contraire que pour le suivi de la ligne, la tête ne va pas suivre une ligne, mais une couleur. En effet, nous ne nous servons pas du module d'Awabot pour suivre les lignes, et nous prenons la position de la couleur comme repère

```
headYaw.val=headYaw.val-
(0.1*(colorDetection.getColor("red").x))
```

4) Dans ce nœud on augmente la vitesse du robot puisqu'on est dans le cas où la ligne est verte. Pour cela, on modifie les paramètres de la vitesse à introduire dans le nœud `MoveBody`. L'interet de la ligne verte serai de la placer sur une ligne droite, donc nous pouvons augmenter la vitesse sans trop élever le risque de perdre la ligne.

Les transitions doivent se programmer en script et permettent d'introduire des conditions, sans quoi, nous ne pourrions pas distinguer quel chemin prendre dans le graphe. Les transitions relient les différents nœuds et permettent d'aller d'un état à un autre. En langage logique, elles seraient l'équivalence d'un *If*.

Les principaux critères que nous avons utilisés dans les transitions, pour passer d'un nœud à l'autre, sont les suivant :

- Ligne visible ?

  ```
  lineDetection.getZone(zone).visible == 1
  ```

  Ou 0 si l'on veut que la ligne ne soit pas visible

- Ligne n'est pas droite ?

  ```
  lineDetection.getZone(zone).x->changed?
  ```

  On compare les valeurs de x pour savoir si la ligne se dévie pour ensuite corriger la position

  La même commande est utilisée pour les lignes rouges

  ```
  colorDetection.getColor("red").x->changed?
  ```

- Couleur visble ?

  ```
  colorDetection.getColor("red").visible==1
  ```

  Ou 0 si on ne veut pas qu'elle soit visible

- Couleur dans la partie inférieure de l'image ?

  ```
  colorDetection.getColor("red").y<-0.2
  ```

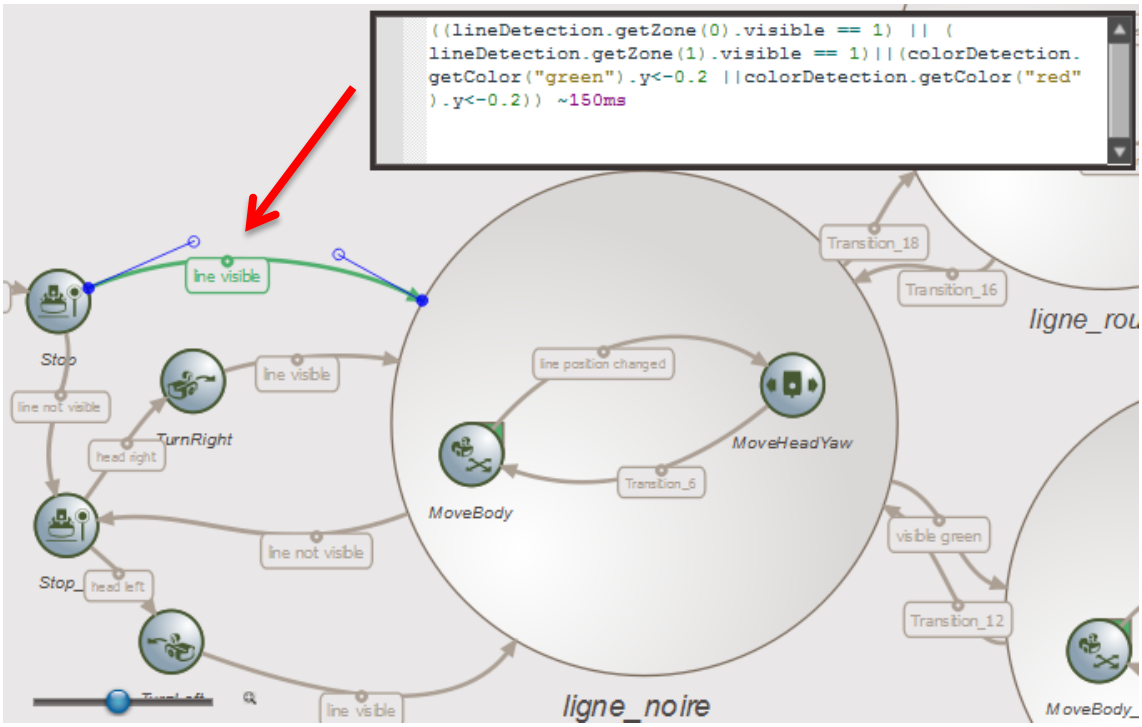
  Nous utilisons ceci pour simuler le comportement du module de détection de ligne d'Awabot. Sans cela, une ligne colorée situé loin à l'horizon sera traitée comme la ligne à suivre.

- Pendant combien de temps ?

  Il suffit de rajouter *~250ms* avec le temps voulue suivie de la condition que l'on veut qui soit vérifiée pendant au moins, ce temps.

Certaines transitions relient des nœuds qui simplement doivent se succéder et il n'est pas nécessaire de mettre des conditions. Par contre, d'autres ont besoins que plusieurs conditions soit vérifiées, ou bien une seule parmi plusieurs. Dans ces cas il faut rajouter *&& (and)* ou bien *|| (or).*

Nous allons voir un exemple de transition où l'on veut passer d'un nœud à un autre uniquement si une ligne est visible dans les zones 0 ou 1 (les plus proches au robot) ou bien si les couleurs rouge ou vertes sont détectées dans la partie inférieure de l'image. De plus, la condition doit être vraie pendant au moins une seconde.



3.7. Solutions de problèmes

3.7.1. Connexion voie router

Étant donné qu'Emox Lab est un prototype, nous avons trouvé quelques soucis pendant le projet. La plupart d'eux sont des problèmes en relation avec la connexion wifi. En fait, c'est une partie laquelle selon M. Faguet, n'est pas encore complètement développée. On doit attendre à la nouvelle version du robot avec le code développé par Awabot même avec laquelle on espère ne pas trouver ces difficultés.

Aujourd'hui, il existe deux façons pour nous connecter au robot : voie ad-hoc ou voie router. Normalement nous avons utilisé la connexion ad-hoc puisque quand nous avons essayé l'autre, nous avons trouvé une connexion pire et moins sûre. Mais de toute façon nous vous détaillons les pas à suivre pour configurer une connexion à travers d'un router. Dans le laboratoire il y en a un qui crée un réseau appelé *default.* D'après les indications de M. Faguet que nous a envoyé par mail, les pas sont les suivants :

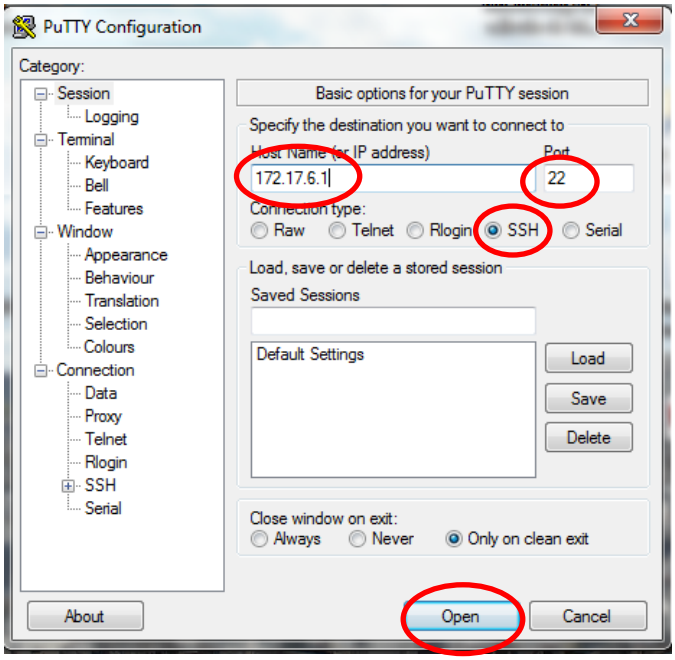
Remarque importante : Avant de manipuler les configurations du robot, on doit être sûrs que la batterie du robot est suffisamment chargée, au moins avec 80% de sa capacité.

1) Télécharger le logiciel *putty*. On se dirige au site: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Et on cherche le fichier putty.exe dans ***Binaries > For Windows on Intel x86.*** Il n'y a pas besoin d'installation, juste double clic.

2) Démarrer le robot et nous connecter à son réseau sous Windows (pas sous Emox Soft).

3) Lancer l'exécutable afin d'obtenir l'ouverture de la fenêtre ci-dessous :

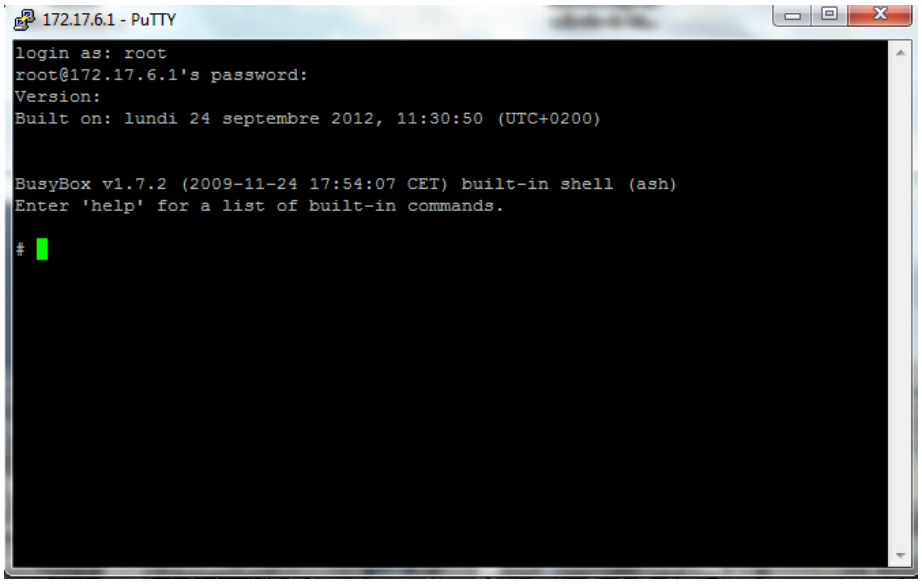


Il faut sélectionner SSH comme type de connexion et écrire :

- Adresse (par défaut en ad-hoc): 172.17.6.1
- Port de communication : 22

Démarrer le programme.

4) Le programme doit ouvrir une console comme la suivante :


172.17.6.1 - PuTTY
login as: root
root@172.17.6.1's password:
Version:
Built on: lundi 24 septembre 2012, 11:30:50 (UTC+0200)
BusyBox v1.7.2 (2009-11-24 17:54:07 CET) built-in shell (ash)
Enter 'help' for a list of built-in commands.
#

On y doit écrire :

- Nom d'utilisateur : **root**
- Mot de passe : **marvell**

Maintenant nous sommes connectés au système interne Linux du robot.

5) Pour configurer la connexion wifi par router, les commandes à taper sont :

```
# cfg set wireless-fav infra:<SSID>::wep ou wap:<key>
# cfg commit
# reboot
```

Dans notre cas :

```
# cfg set wireless-fav infra:default
# cfg commit
# reboot
```

6) Les leds hauts du robot seront étendues et alors on doit redémarrer le robot.

Peut-être on veut **remettre les configurations initiales du wifi** pour nous connecter voie ad-hoc car on ne veut plus se servir du router. Les pas à suivre sont les mêmes qu'on a suivi ci-dessus, mais au pas 5 nous devons taper :

```
# cfg set wireless-fav infra:<>::wep:<>
# cfg commit
# reboot
```

3.7.2. Des autres problèmes trouvés

Après avoir changé la configuration du réseau pour nous connecter à travers du router, nous avons testé que la connexion n'été pas plus performante et nous avons plus de pertes de connexions, raison pour laquelle nous avons suivi les pas nécessaires pour retourner à la configuration initiale. Après cette démarche, nous n'avons pas réussi à nous connecter. Malgré que le réseau ad-hoc a été créée et que nous trouvons le robot depuis le logiciel, au moment de la connexion sur le coin inferieur droit on trouvait *Emox_Cristal_0005/Concecting....*

Si on trouve ce souci, une solution possible est de reflasher le firmware du robot. Les pas qu'on doit suivre sont les suivants :

Remarque importante : Avant de manipuler les configurations du robot, on doit être sûrs que la batterie du robot est suffisamment chargée, au moins avec 80% de sa capacité.

1) Téléchargez les logiciels *putty* et *pscp*. On se dirige au site:

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Et on cherche les fichiers *putty.exe* et *pscp.exe* dans ***Binaries > For Windows on Intel x86.*** Il n'y a pas besoin d'installation, juste double clic.

2) Mettez le fichier du firmware à l'emplacement où on a gardé pscp.exe. Ce que nous avons est nommé : *rootfs.arm_nofpu.squashfs_1_22.*

3) Ouvrez une invite de commande et placez-vous dans le dossier où on a mis le pscp.exe et le firmware. Pour cela, tapez *cd* et ajoutez après l'adresse. Vous pouvez la copier depuis le dossier Windows et la mettre dans l'invite de commande en faisant bouton droit de la souris et *Coller.*

4) On doit étendre le router, allumer le robot et nous connecter à son réseau sous Windows.

5) Puis exécutez :

```
pscp.exe -scp <NomDuFirmware> root@172.17.6.1:/tmp
```

Dans notre cas :

```
pscp.exe -scp rootfs.arm_nofpu.squashfs_1_22
                                        root@172.17.6.1:/tmp
root@172.17.6.1's password:
```

Le mot de passe sera demandé : *marvell*

Le programme installera le firmware. Attendez au message :

```
rootfs.arm_nofpu.squashfs | 2688 kB | 134.4 kB/s | ETA:
00:00:00 | 100%
```

6) Connectez-vous au robot par putty.exe (voir pas *3* sous le titre *Connexion voie router*).

7) Dans la console de putty, exécutez :

```
flashcp <NomDuFirmware> /dev/mtd1
```

soit :

```
flashcp  rootfs.arm_nofpu.squashfs_1_22 /dev/mtd1
```

8) Attendez que le prompt soit disponible (attendez au symbole #), puis tapez `reboot` et finalement redémarrez le robot.

4. Conclusion

Grâce à ce projet nous avons eu l'occasion de connaître deux robots différents, les diverses façons existantes pour les contrôler et, en plus, nous avons appris des nouveaux langages de programmation.

Nous avons établi les bases pour la continuation de ces projets au sein du laboratoire de Mécatronique et Robotique. Par rapport à Nao, nous avons déjà fait la recherche des librairies et applications existantes pour arriver à une solution. Nous avons d'autre part étudié les multiples façons de les intégrer lesquelles sont toutes détaillées dans le rapport.

Pour la partie d'Emox, nous avons été les élèves chargés de l'accueillir, de l'étudier et de découvrir et apprendre son fonctionnement et le langage associé. En plus, nous avons expliqué les possibles soucis avec ce robot prototype et les solutions connues jusqu'au présent. Nous espérons que la nouvelle version du robot sera plus performante, plus fiable et que le système embarqué, cette fois développé par Awabot même, permettra une programmation plus complète et accessible.

BIBLIOGRAPHIE

Introduction

[1] http://developer.aldebaran-robotics.com
[2] http://www.youtube.com/watch?v=4eBqVn3uS5w
[3] http://searchfortechnology.com/emox-intelligent-entertainment-robot
[4] http://awabot-redmine.herokuapp.com/

NAO

Installation des outils et des librairies nécessaires pour la lecture avec NAO sur Linux Ubuntu

[5] http://www.aldebaran-robotics.com/documentation/dev/cpp/install_guide.html
[6] http://old-releases.ubuntu.com/releases/
[7] http://code.google.com/p/qrdecoder/

Les codes étudiés

[8] http://qrworld.wordpress.com/2011/09/26/qr-codes-versus-data-matrix/
[9] http://datamatrix.kaywa.com/
[10] http://www.barcodeisland.com
[11] http://www.unitaglive.com/qrcode
[12] Athanasia LOULOUDI, Ahmed MOSALLAM, Naresh MARTURI, Pieter JANSE and Victor HERNANDEZ. http://ebookbrowse.com/sais2010-ipw-paper-pdf-d79389844. School of Science and Technology. Örebro University, Sweden

Les librairies et les applications étudiées

- qrdecoder :

[13] http://code.google.com/p/qrdecoder/

- PyQT4, PIL et SIP

[14] http://www.riverbankcomputing.co.uk/software/pyqt/intro
[15] http://www.riverbankcomputing.co.uk/software/sip/intro
[16] http://www.pythonware.com/products/pil/
[17] http://www.pythonware.com/library/pil/handbook/image.htm

- ZBar

[18] http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=HOWTO:_Scan_images_using_the_API
[19] http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=HOWTO:_Scan_video_using_the_Processor
[20] http://stackoverflow.com/questions/9190687/using-OpenCV-instead-of-imagemagick-in-ZBar-qr-code-decoder
[21] http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=ZBar_Wiki

- OpenCV

[22] Site officiel d'OpenCV : http://docs.OpenCV.org/
[23] http://OpenCV.willowgarage.com/wiki/
[24] http://www.mattmontag.com/development/notes-on-using-OpenCV-2-3-with-visual-studio-2010
[25] http://docs.OpenCV.org/modules/core/doc/intro.html

- Libdmtx :

[26] http://www.libdmtx.org/

- ImageMagick :

[27] http://imagemagick.org

- Autres librairies étudiées

[28] ZXing http://code.google.com/p/zxing/
[29] Libdecodeqr : https://github.com/josephholsten/libdecodeqr

Intégration des applications sur NAO

[30] https://developer.aldebaran-robotics.com/home/
[31] http://www.aldebaran-robotics.com/documentation/software/choregraphe/index.html

EMOX

Description technique

[32] http://fr.wikipedia.org/wiki/Pont_en_H
[33] http://es.wikipedia.org/wiki/Puente_H_(electronica)
[34] http://www.dema.net/pdf/sharp/GP2Y0A41SK0F.pdf
[35] http://www.alldatasheet.com/view.jsp?Searchword=GP2Y0) :
[36] http://www.pololu.com/file/0J507/HD-1581HB.pdf)
[37] http://datasheet.octopart.com/HN-GH12-1634T-R-Jameco-Reliapro-datasheet-7274139.pdf
[38] http://pdf1.alldatasheet.com/datasheet-pdf/view/42773/SHARP/GP1A53HR.html
[39] http://www.datasheetdir.com/OV530-B49+download
[40] http://www.arduino.cc/fr/
[41] http://www.jameco.com/Jameco/Products/ProdImag/161382.jpg
[42] FAGUET, Thomas. Documentation d'EMOX : *Doc Hardware.pdf.* Version 1 du 12/06/2012. Awabot Robotics.

Installation et connexion

[43] Documentation d'EMOX : *Manuel d'utilisation.pdf.* Awabot Robotics. 2012.

Programmation du robot

[44] Documentation d'EMOX : *Document de programmation graphique V1.pdf.* Awabot Robotics. Version 1.0.0. Septembre 2012.
[45] Documentation d'EMOX : *Emox_SDK.pdf.* Awabot Robotics . 18 juin 2012.

Suivi d'une ligne

[46] http://botbench.com/blog/2011/09/16/exposed-lego-colour-sensor/

Détection de la couleur

[47] http://awabot-redmine.herokuapp.com/boards/5/topics/4

[48] play.google.com/store/apps/details?id=com.mobialia.colordetector

[49] play.google.com/store/apps/details?id=com.loomatix.colorgrab

Solutions de problèmes

[50] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html